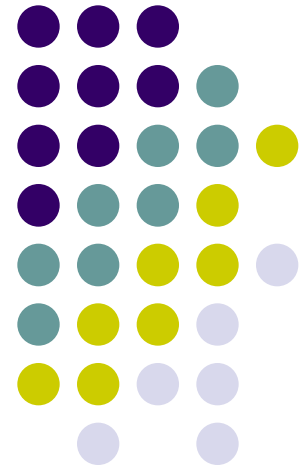
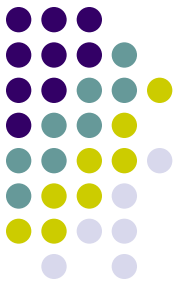


Øvingsforelesning 8

Bellman-Ford og Floyd-Warshall

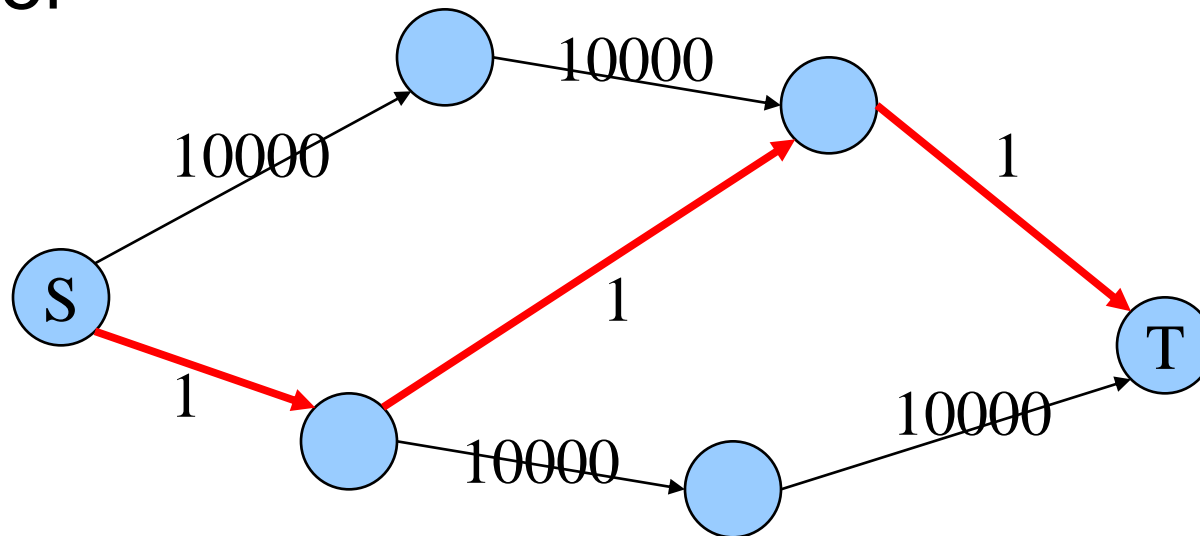
Kristian Veøy
veoy@stud.ntnu.no





Korteste vei - repetisjon

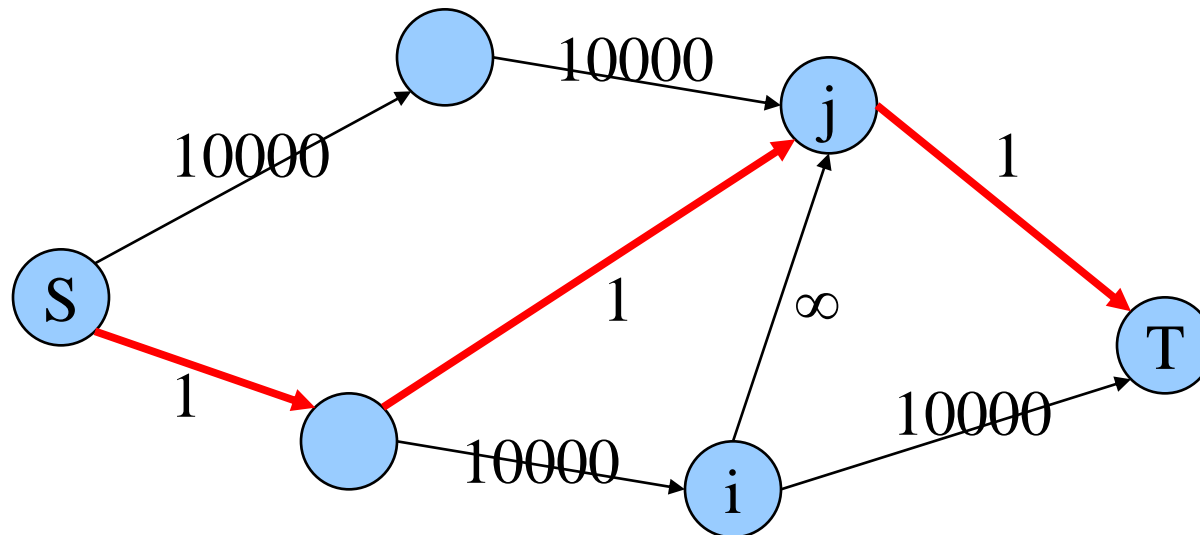
- Vi har en vektet graf
- Vi vil finne en minimal vekt-sum sti imellom noder

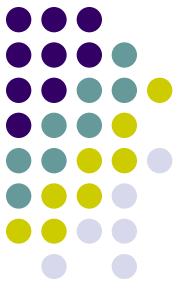




Korteste vei - repetisjon

- Noder som ikke kan nå hverandre har avstand ∞ ($w_{ij} = \infty$)

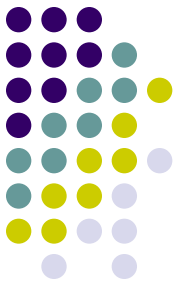




Korteste vei - repetisjon

- Noder har avstand 0 til seg selv





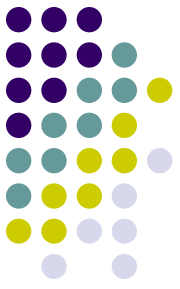
Korteste vei - repetisjon

- Vi vil ha sykelfrie stier (simple paths)



Korteste vei - repetisjon

- Vi vil ha sykelfrie stier (simple paths)
 - Om vi har en negativ sykel har stien vektsum $-\infty$



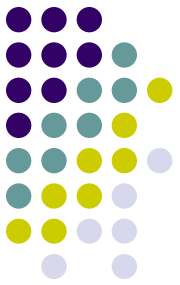
Korteste vei - repetisjon

- Vi vil ha sykelfrie stier (simple paths)
 - Om vi har en negativ sykel har stien vektsum $-\infty$
 - Om vi har en positiv sykel blir stien kortere av å fjerne den



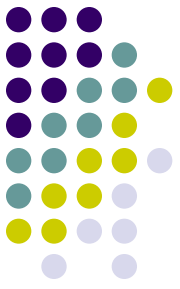
Korteste vei - repetisjon

- Vi vil ha sykelfrie stier (simple paths)
 - Om vi har en negativ sykel har stien vektsum $-\infty$
 - Om vi har en positiv sykel blir stien kortere av å fjerne den
 - Om vi har en 0-vekt sykel kan vi fjerne den uten at det påvirker vekten til stien



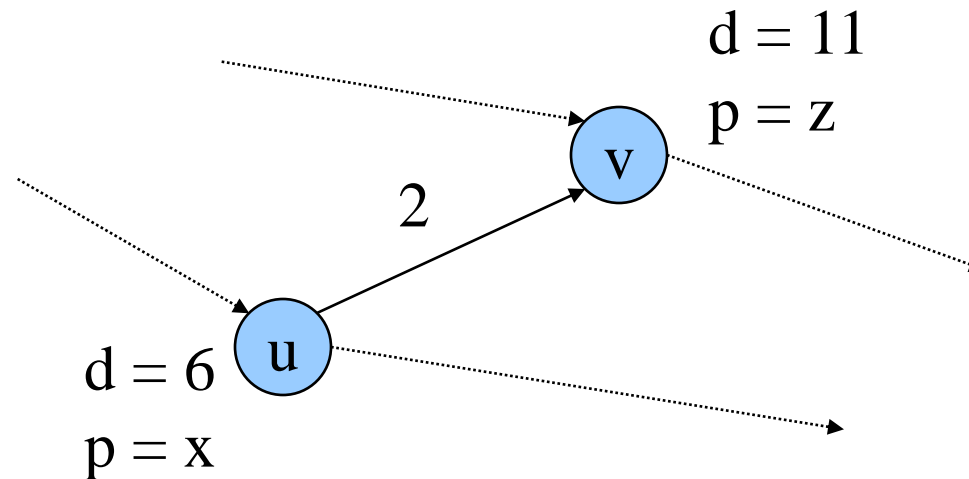
Korteste vei - repetisjon

- Vi vil ha sykelfrie stier (simple paths)
 - Om vi har en negativ sykel har stien vektsum $-\infty$
 - Om vi har en positiv sykel blir stien kortere av å fjerne den
 - Om vi har en 0-vekt sykel kan vi fjerne den uten at det påvirker vekten til stien
- Vi tar kun for oss sykelfrie stier



Korteste vei - repetisjon

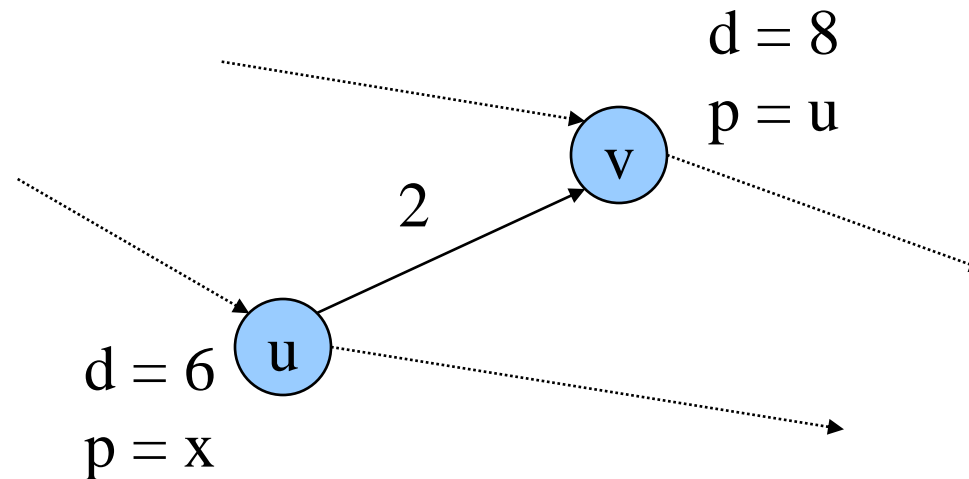
```
def relax(u, v):  
    if d[v] > d[u] + weight[u][v]:  
        d[v] = d[u] + weight[u][v]  
        parent[v] = u
```





Korteste vei - repetisjon

```
def relax(u, v):  
    if d[v] > d[u] + weight[u][v]:  
        d[v] = d[u] + weight[u][v]  
        parent[v] = u
```





Dijkstras algoritme - repetisjon

- Korteste vei en-til-alle uten negative kanter
- Velg node med kortest avstand og oppdater estimer



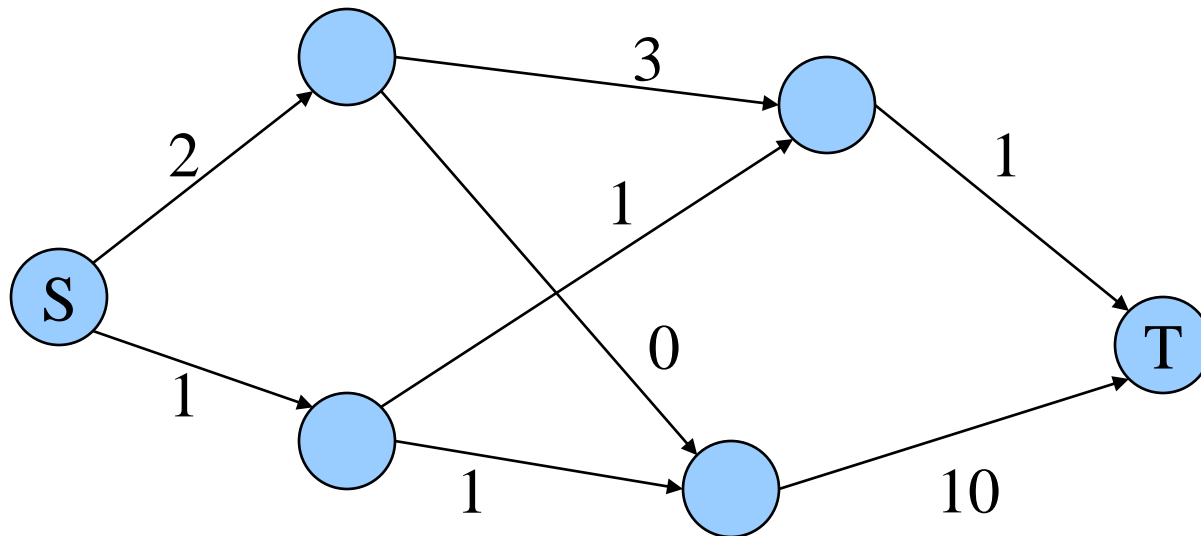
Dijkstras algoritme - repetisjon

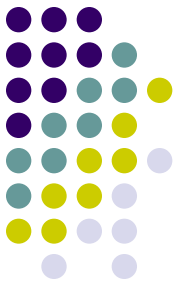
- Korteste vei en-til-alle uten negative kanter
- Velg node med kortest avstand og oppdater estimer

- Intuitivt:
 - Det laveste estimatet kan umulig forbedres ettersom vi ikke har negative kanter

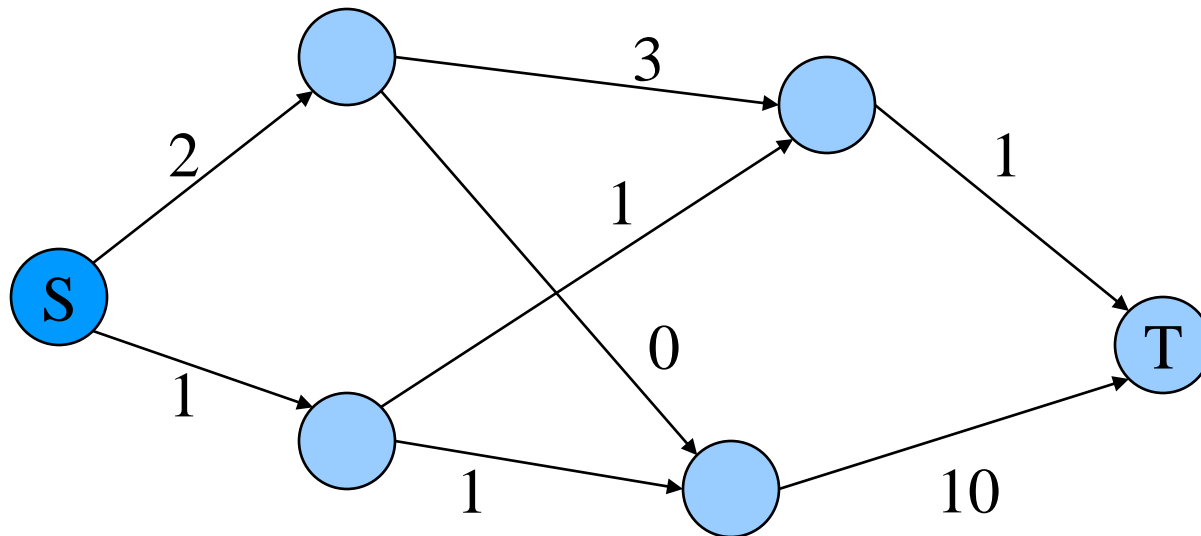


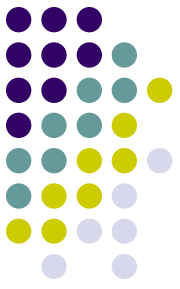
Dijkstras algoritme - repetisjon



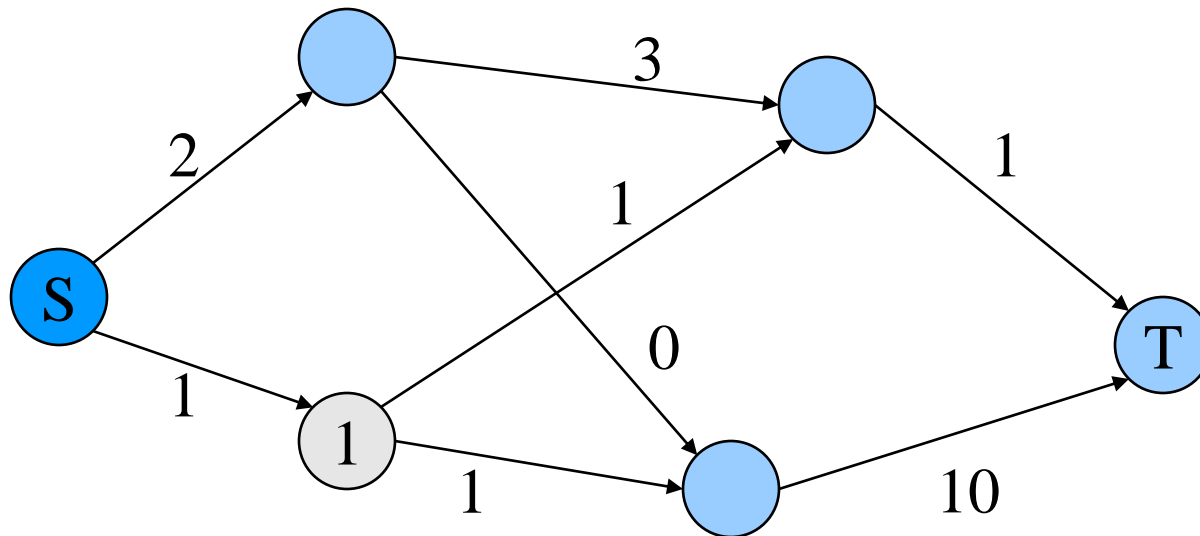


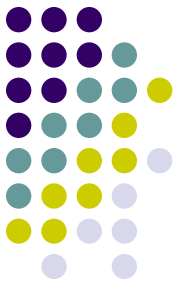
Dijkstras algoritme - repetisjon



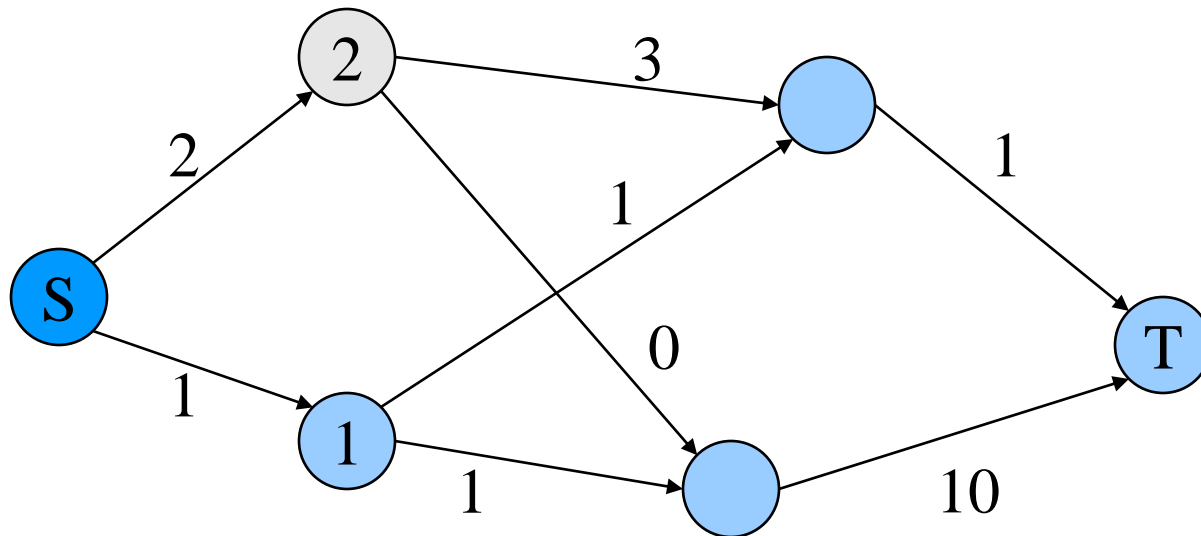


Dijkstras algoritme - repetisjon



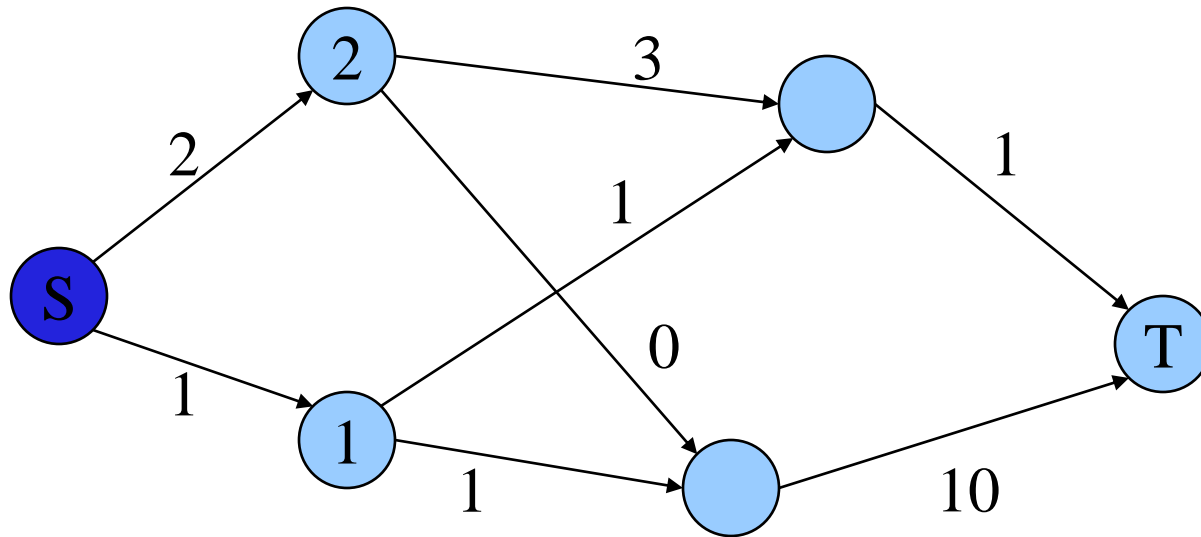


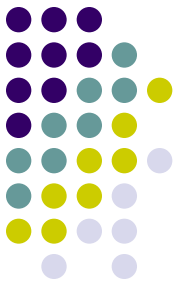
Dijkstras algoritme - repetisjon



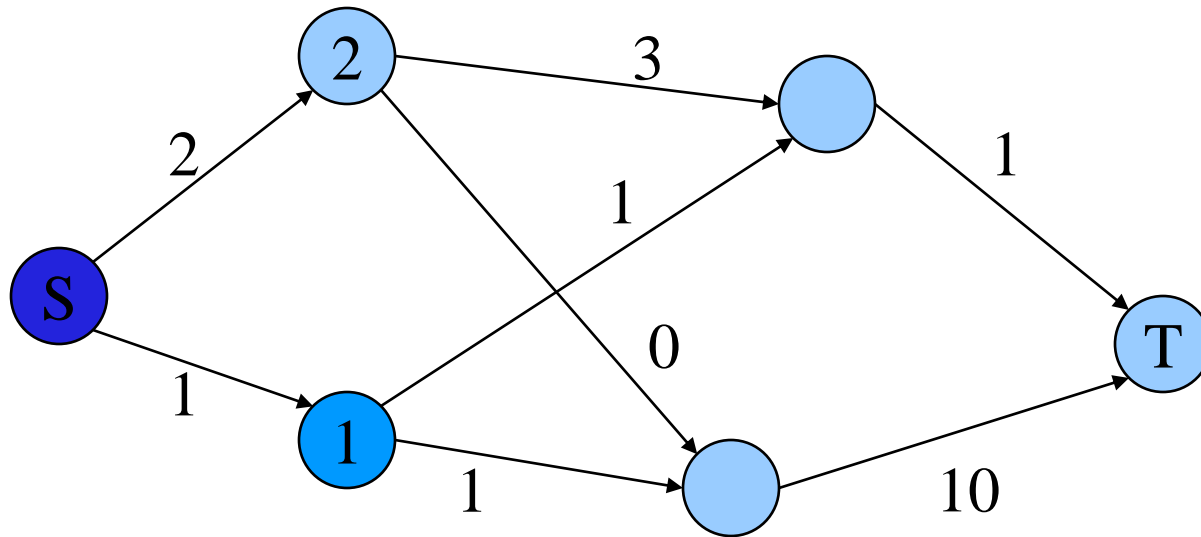


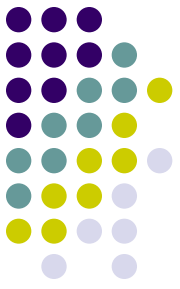
Dijkstras algoritme - repetisjon



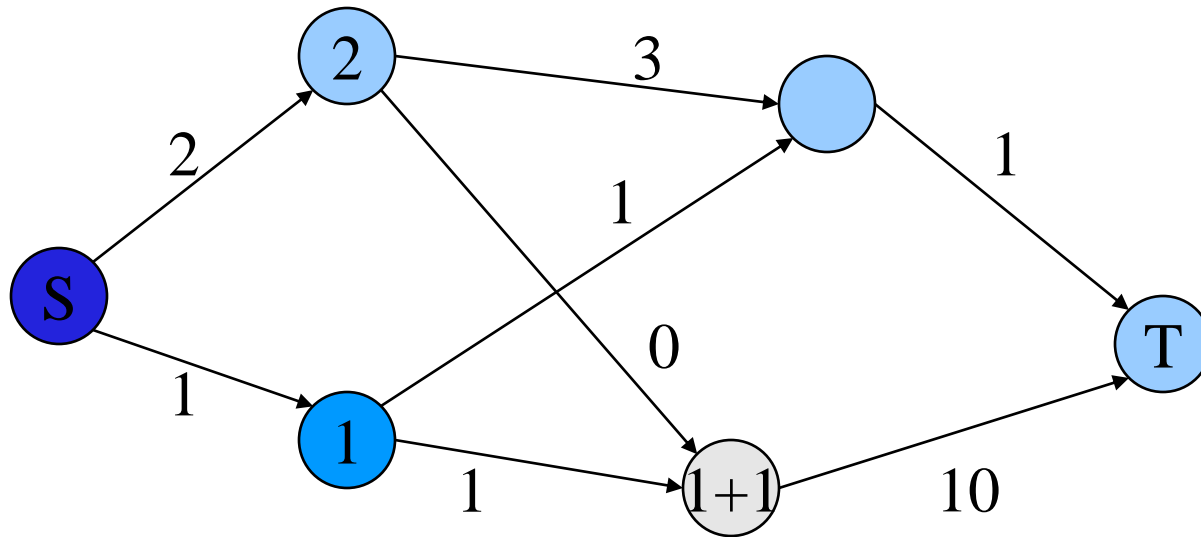


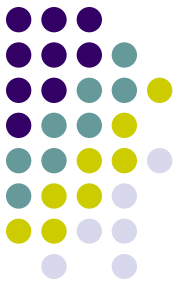
Dijkstras algoritme - repetisjon



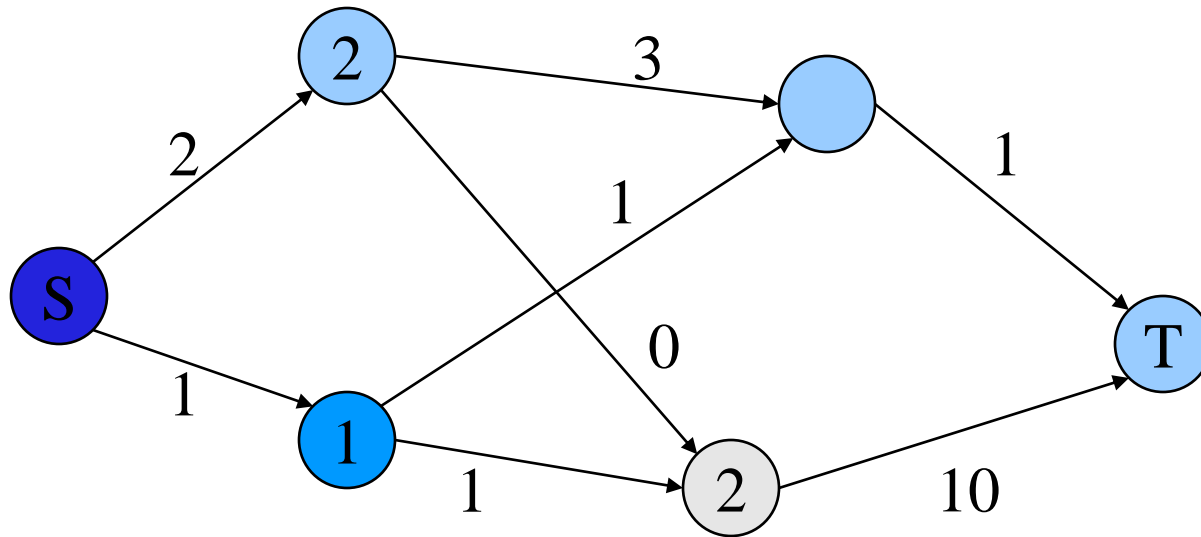


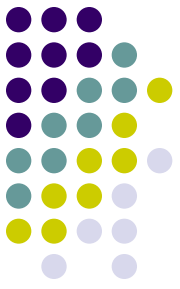
Dijkstras algoritme - repetisjon



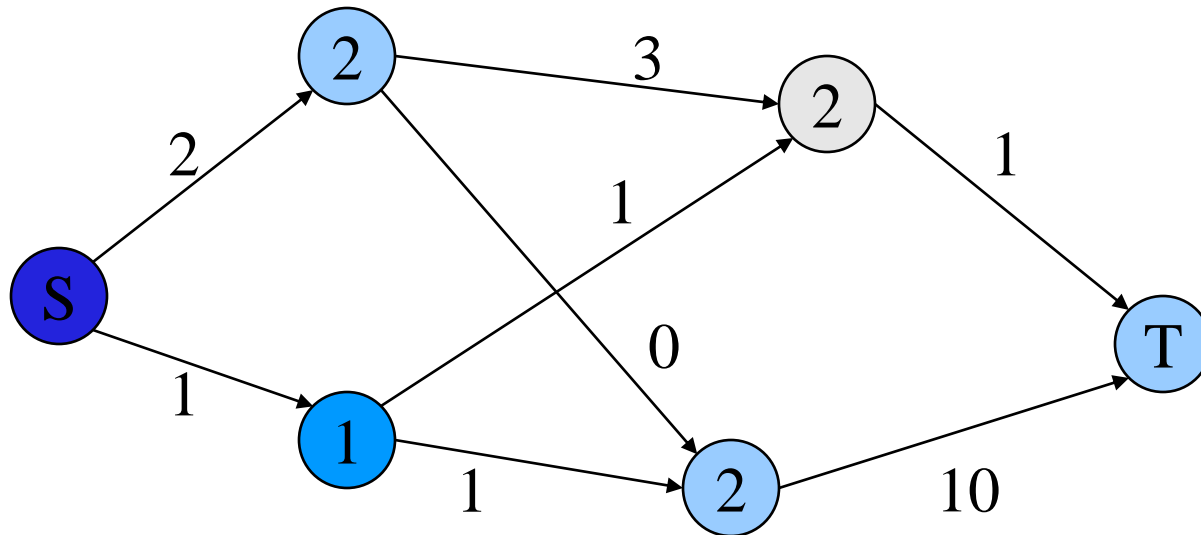


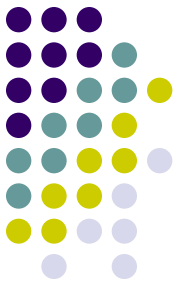
Dijkstras algoritme - repetisjon



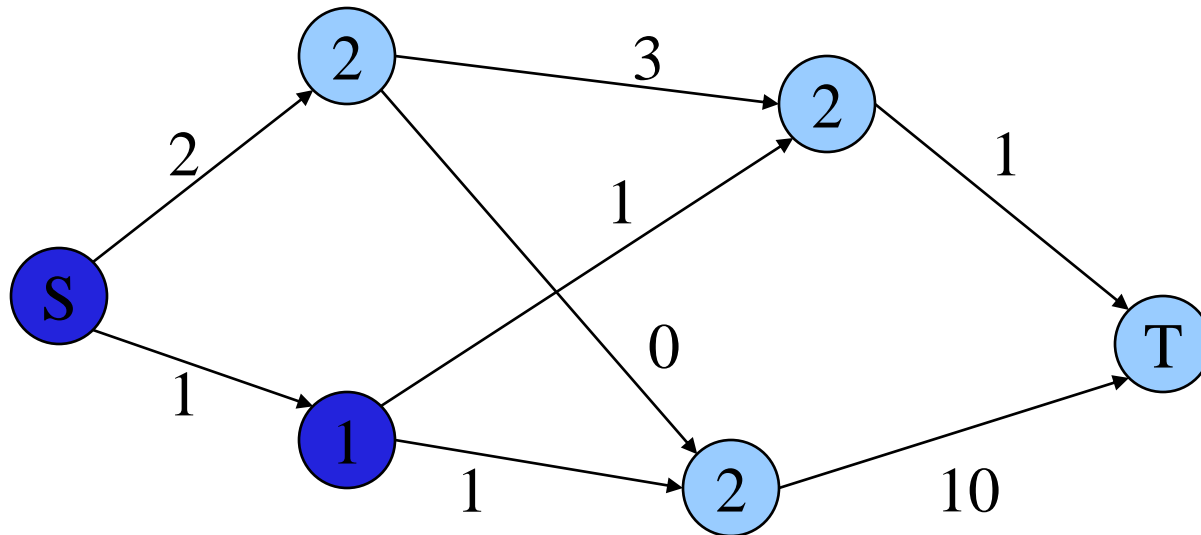


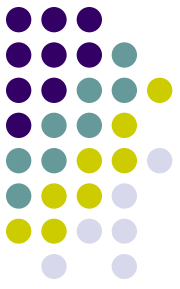
Dijkstras algoritme - repetisjon





Dijkstras algoritme - repetisjon





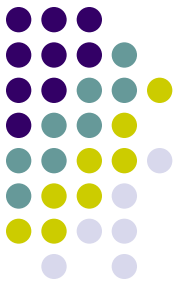
Dijkstras algoritme - repetisjon

- Kjøretid avhengig av implementasjon:
 - Array: $O(V^2)$ fint for dense grafer
 - Heap: $O(E \lg V)$ fint for sparse grafer



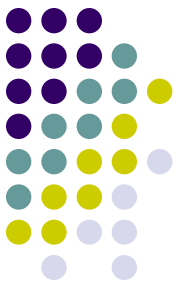
Bellman-Ford

- Korteste vei en-til-alle
- Men, her kan vi ha negative kanter
 - Da kan negative sykler oppstå



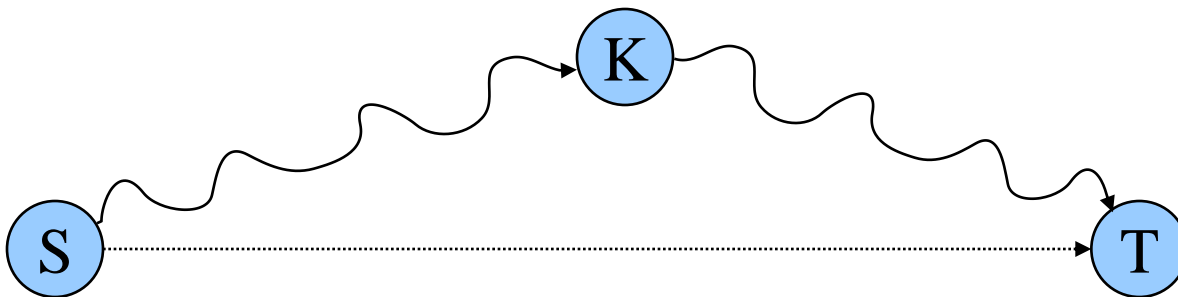
Bellman-Ford

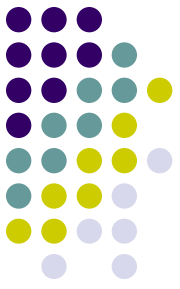
- Den korteste veien kan maks gå innom alle nodene; ellers har vi en sykel



Bellman-Ford

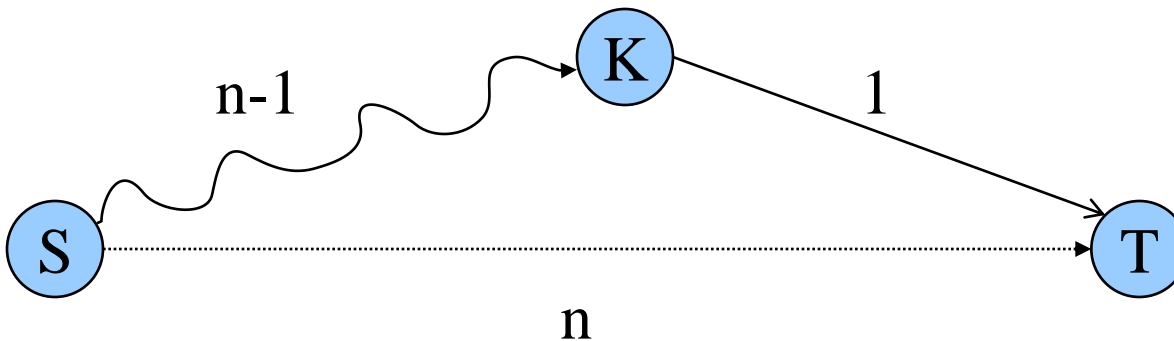
- Den korteste veien kan maks gå innom alle nodene; ellers har vi en sykel
- Vi kan utnytte at en dekomponert korteste vei består av korteste veier:

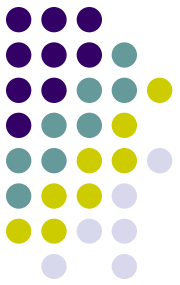




Bellman-Ford

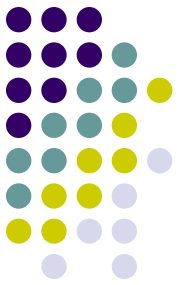
- Den korteste veien kan maks gå innom alle nodene; ellers har vi en sykel
- Vi kan utnytte at en dekomponert korteste vei består av korteste veier:





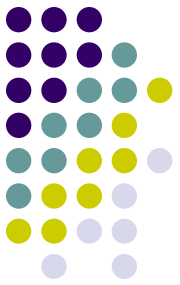
Bellman-Ford

- Hvis vi først finner alle veier som bruker i kanter, kan vi utvide disse til å bruke $i+1$ kanter
 - Sjekk for alle kanter om vi kan gi en kortere vei



Bellman-Ford

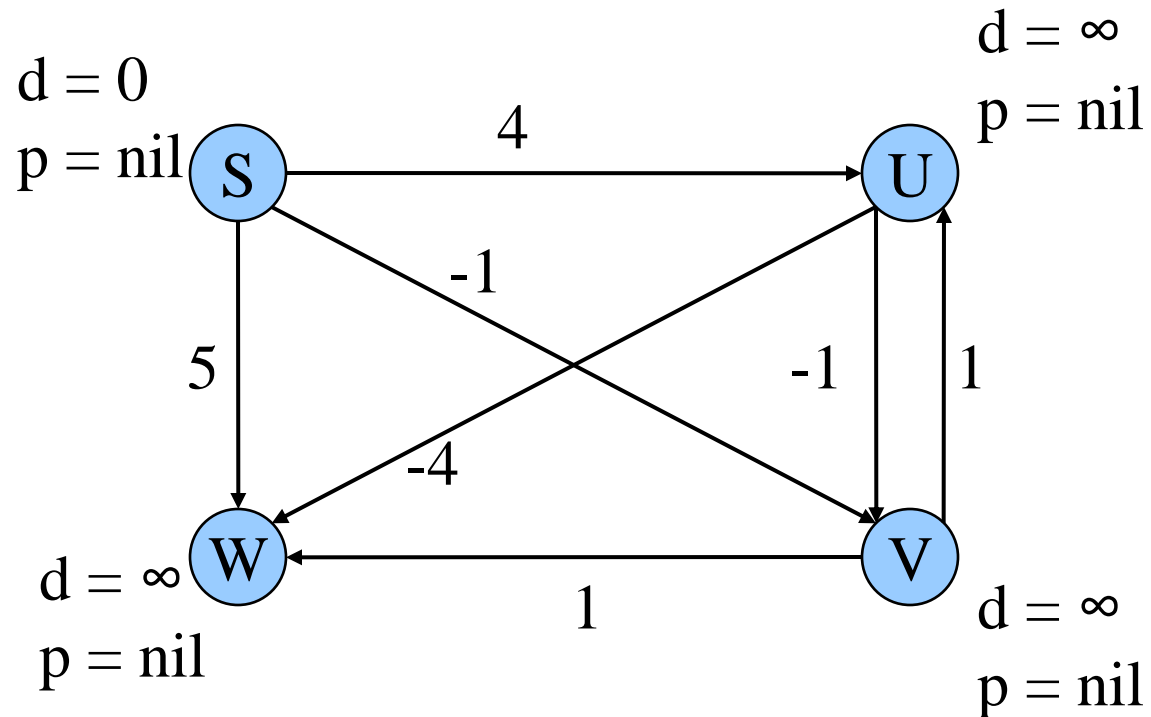
- Hvis vi først finner alle veier som bruker i kanter, kan vi utvide disse til å bruke $i+1$ kanter
 - Sjekk for alle kanter om vi kan gi en kortere vei
- Begynn med stier av lengde 1, og utvid til lengde $V - 1$

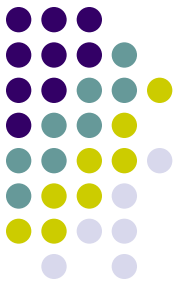


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

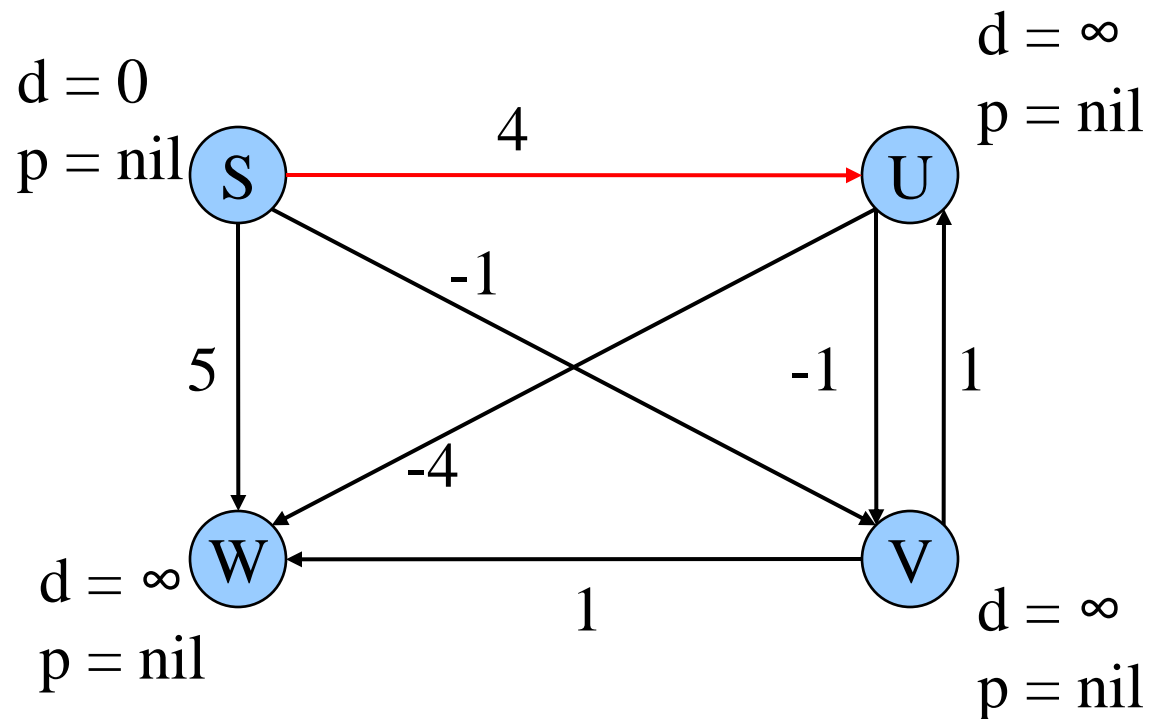




Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

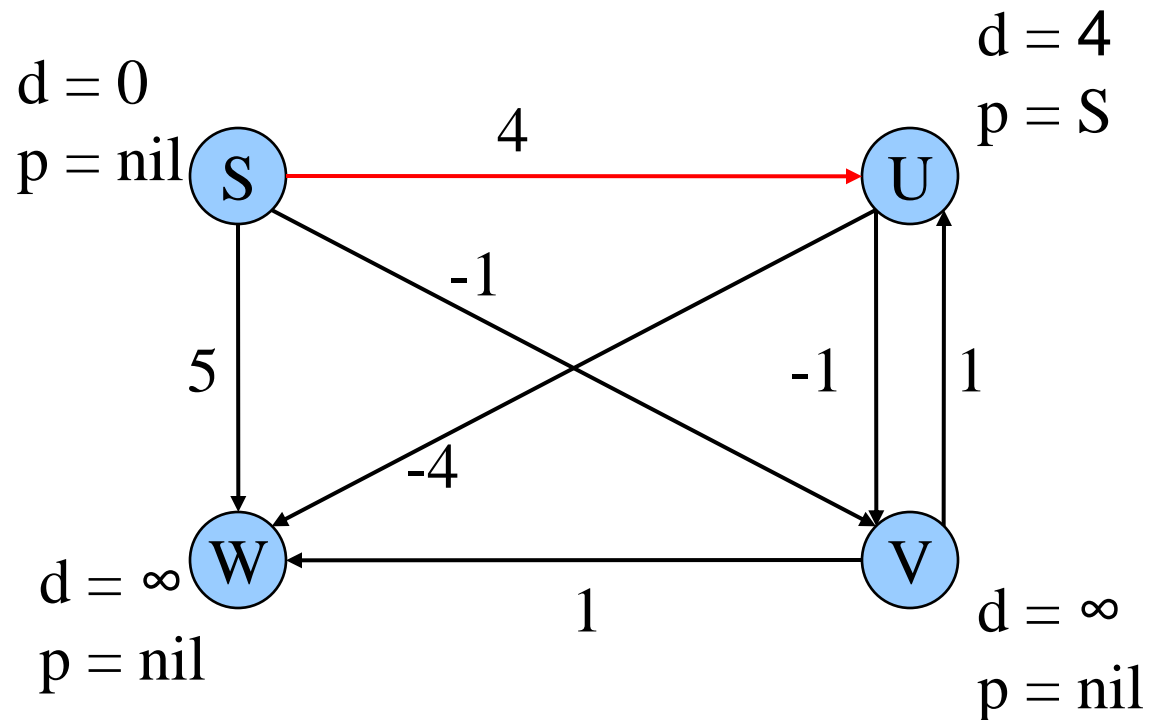




Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

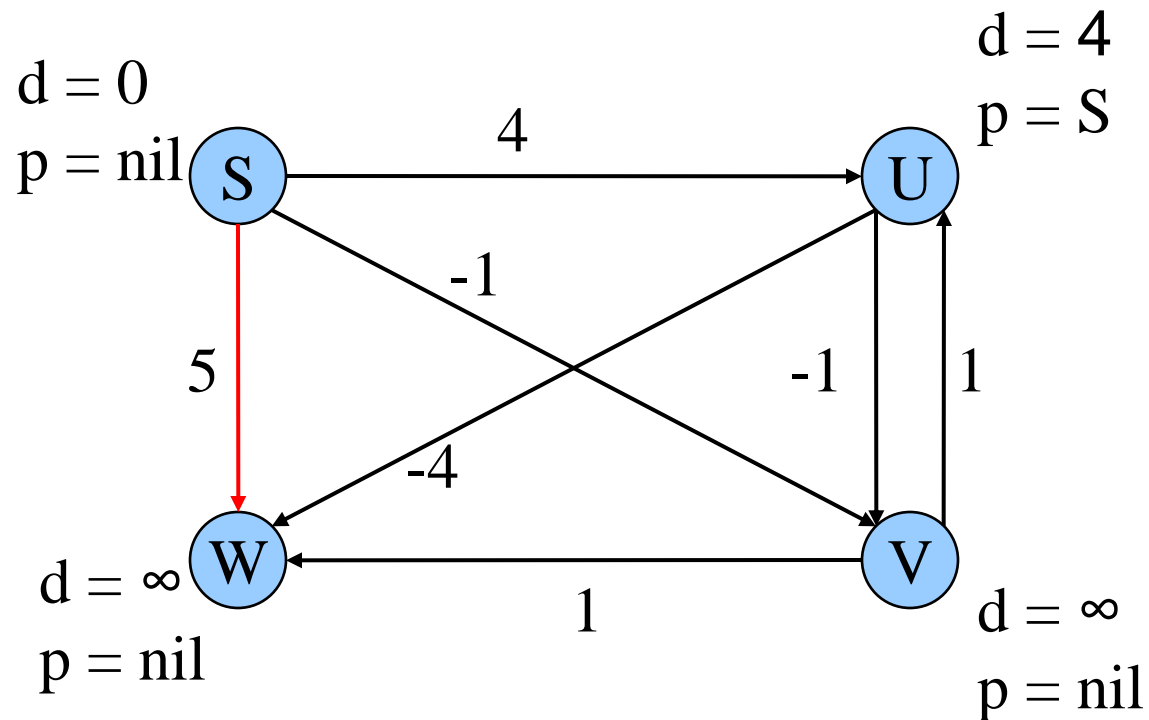


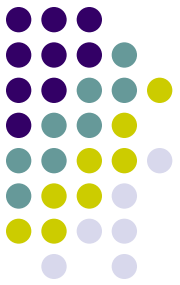


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=1$

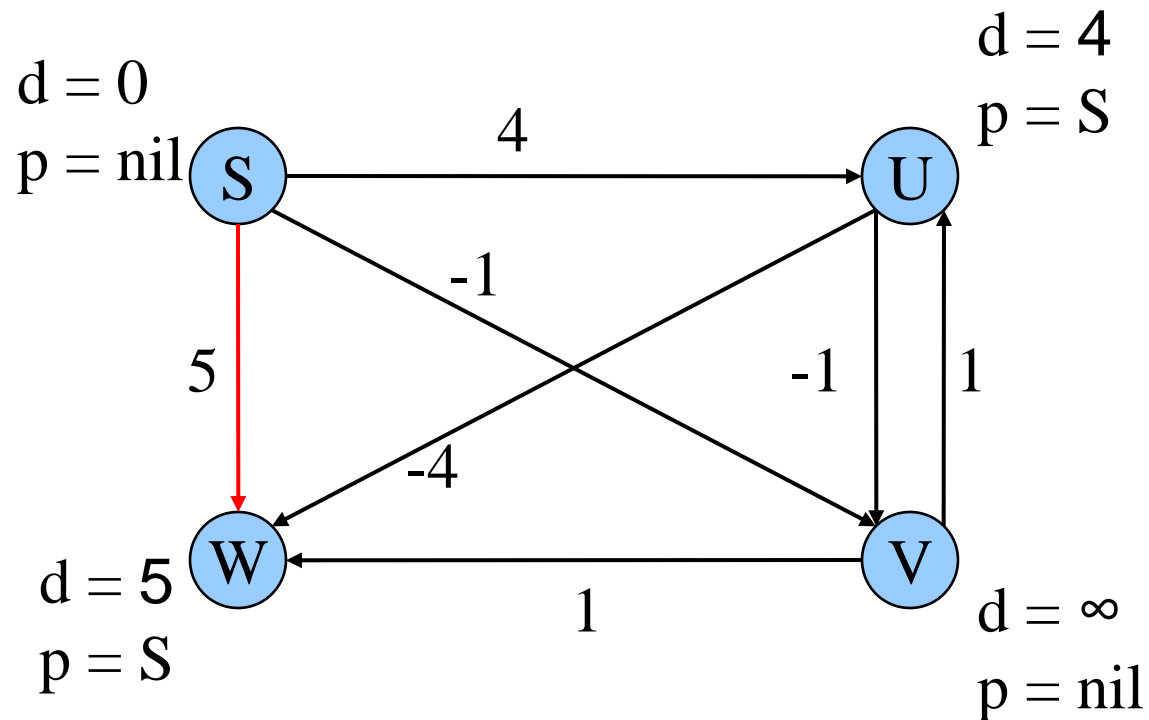


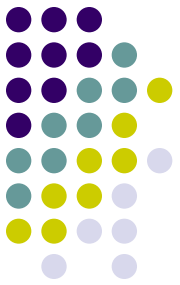


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

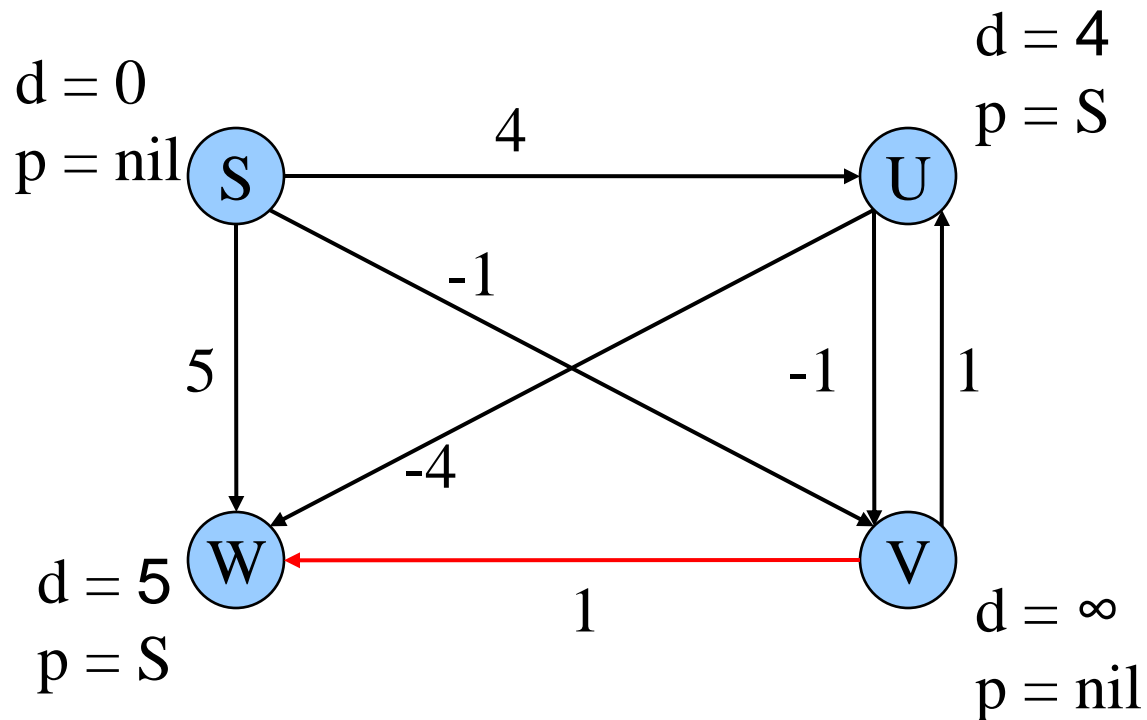


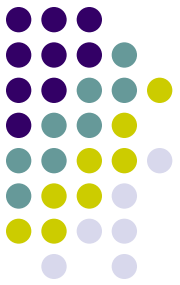


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=1$

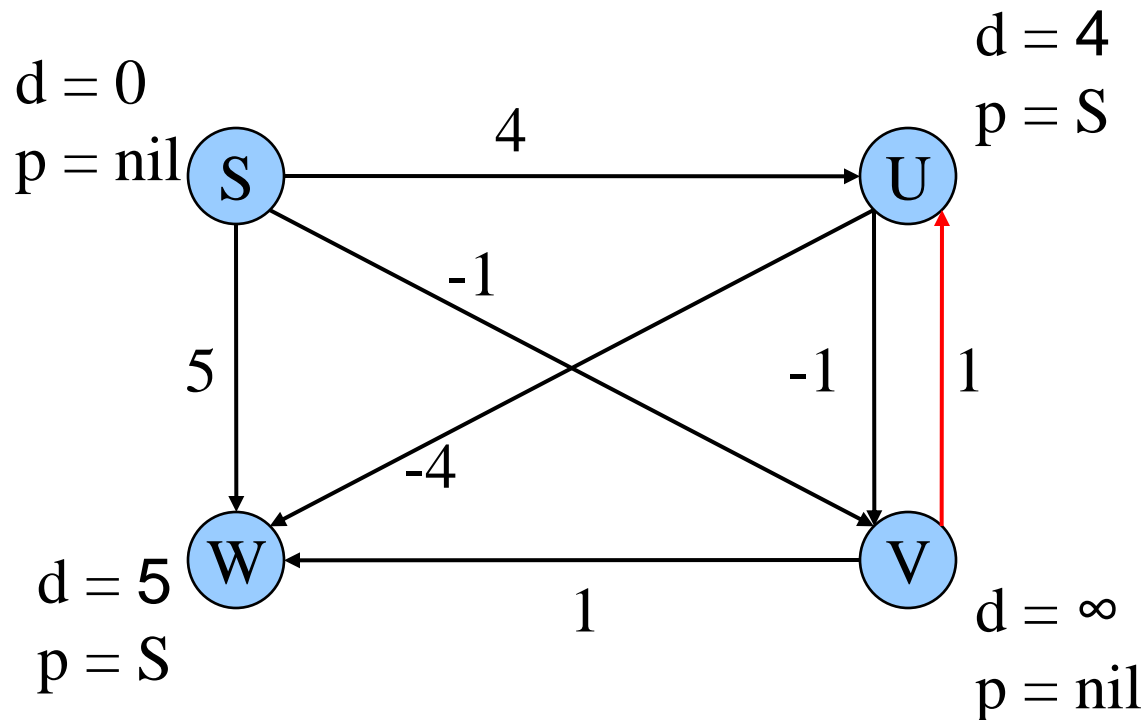


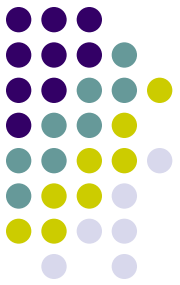


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

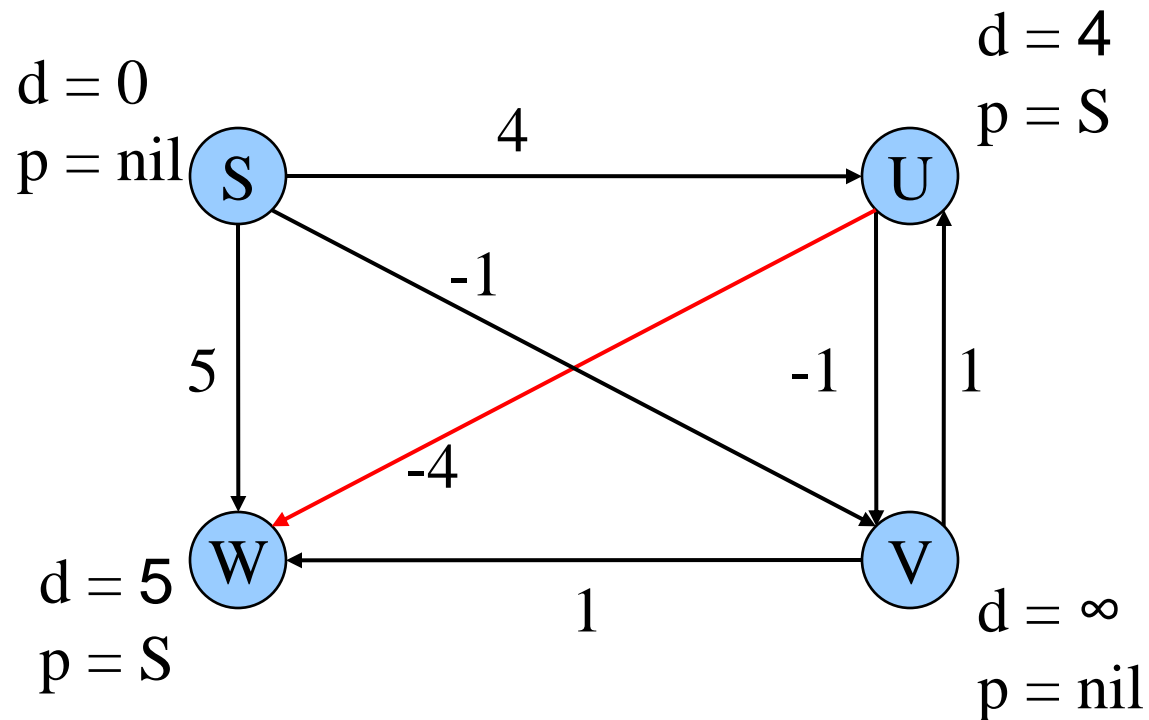


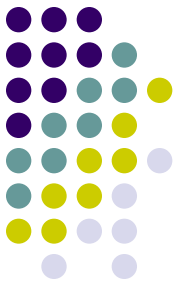


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

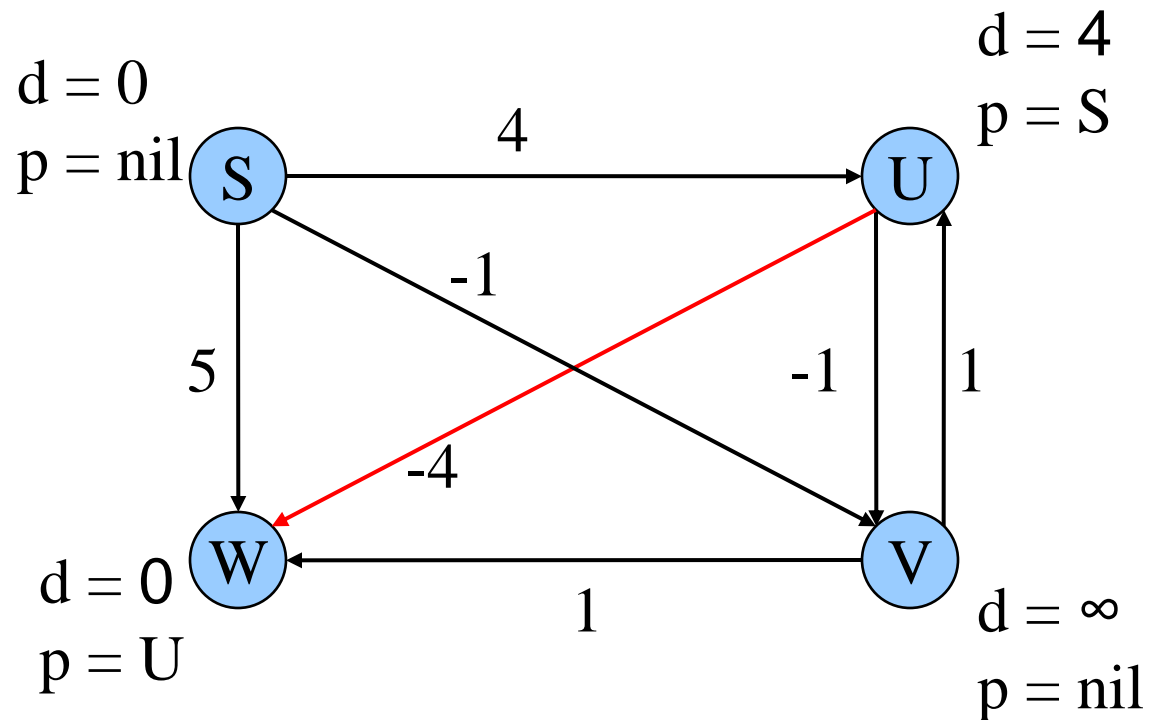


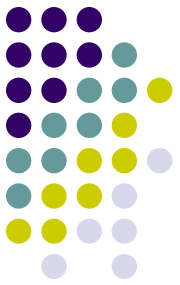


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=1$

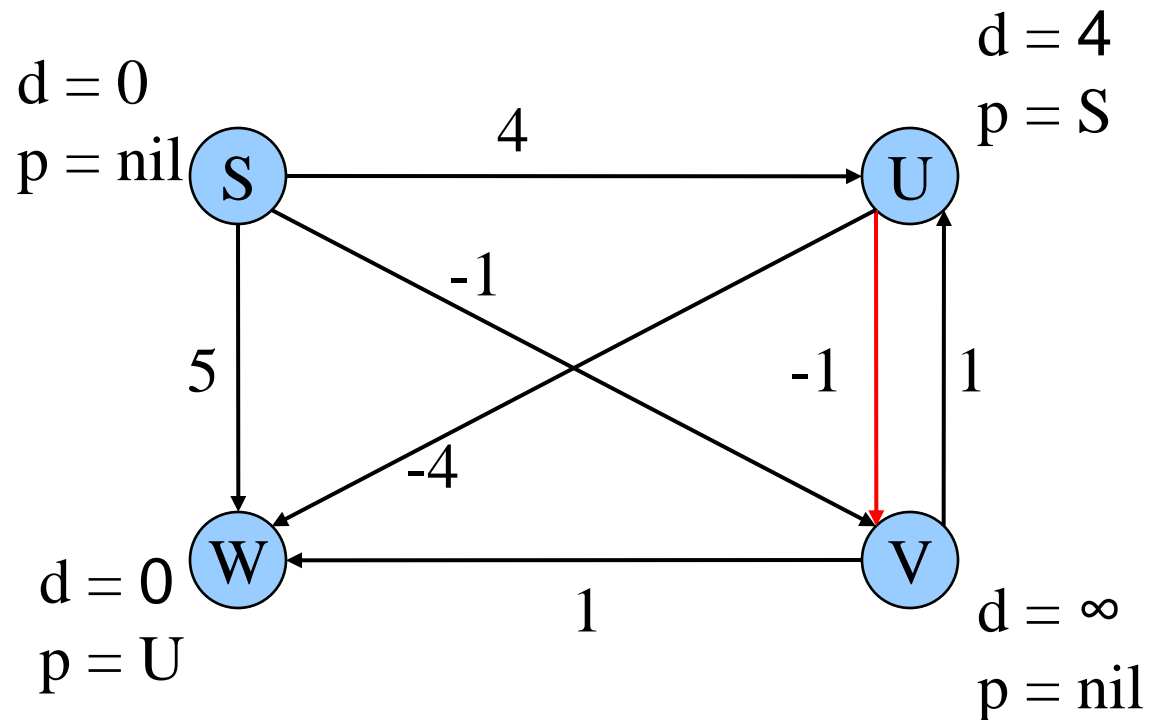




Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=1$

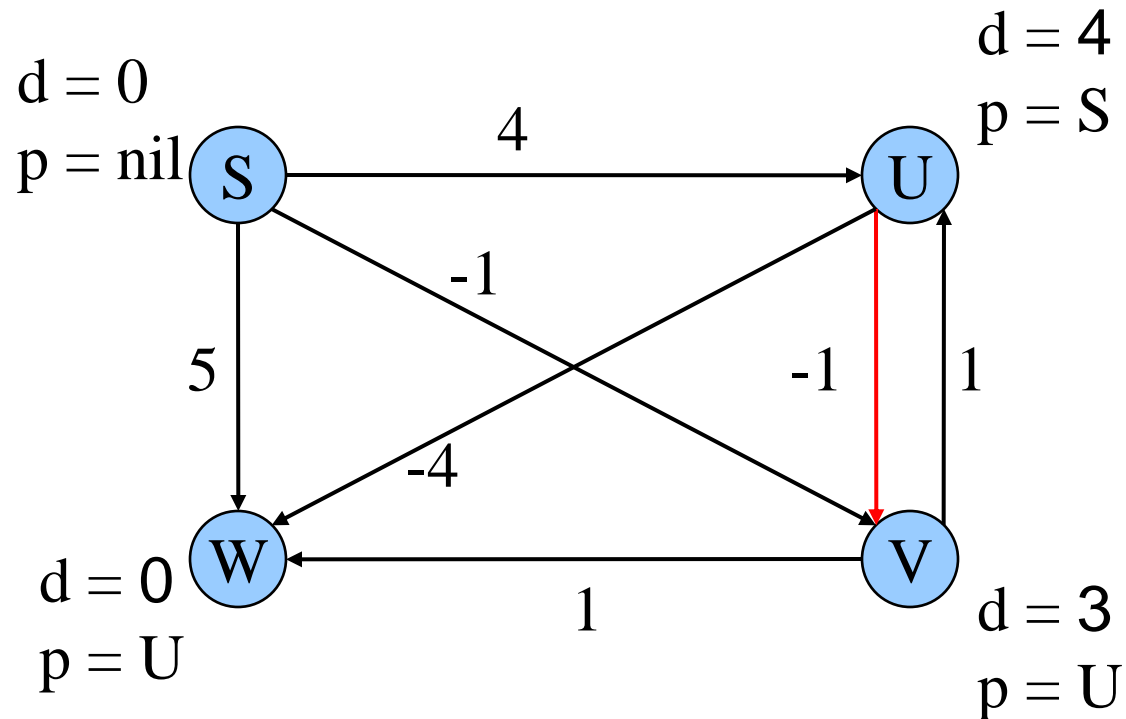


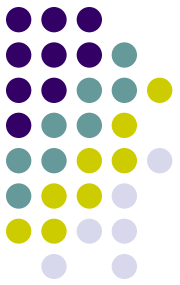


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

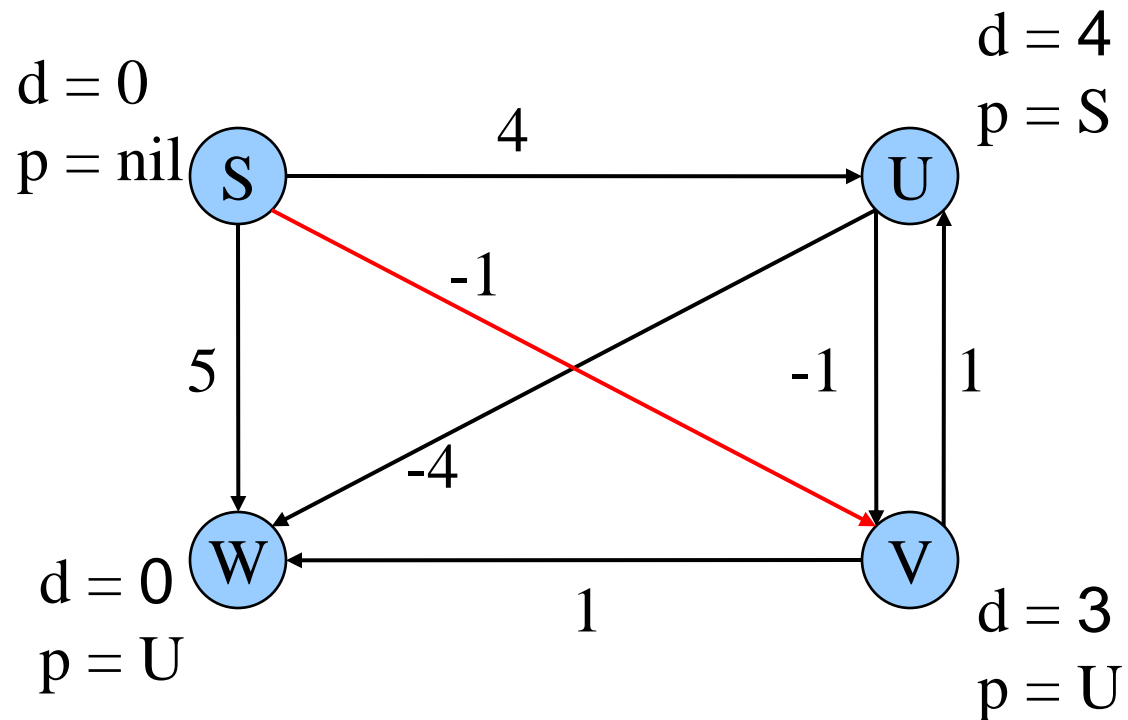




Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=1$

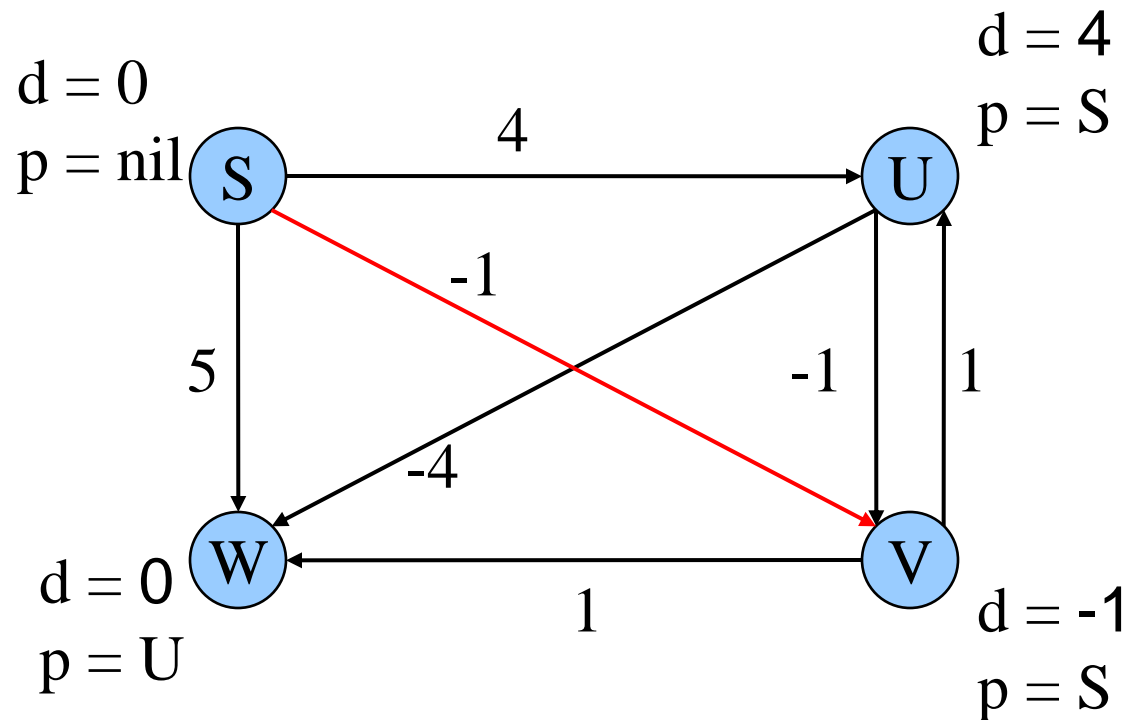


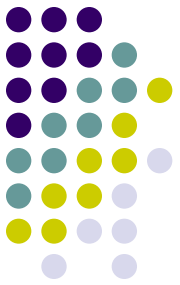


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=1$

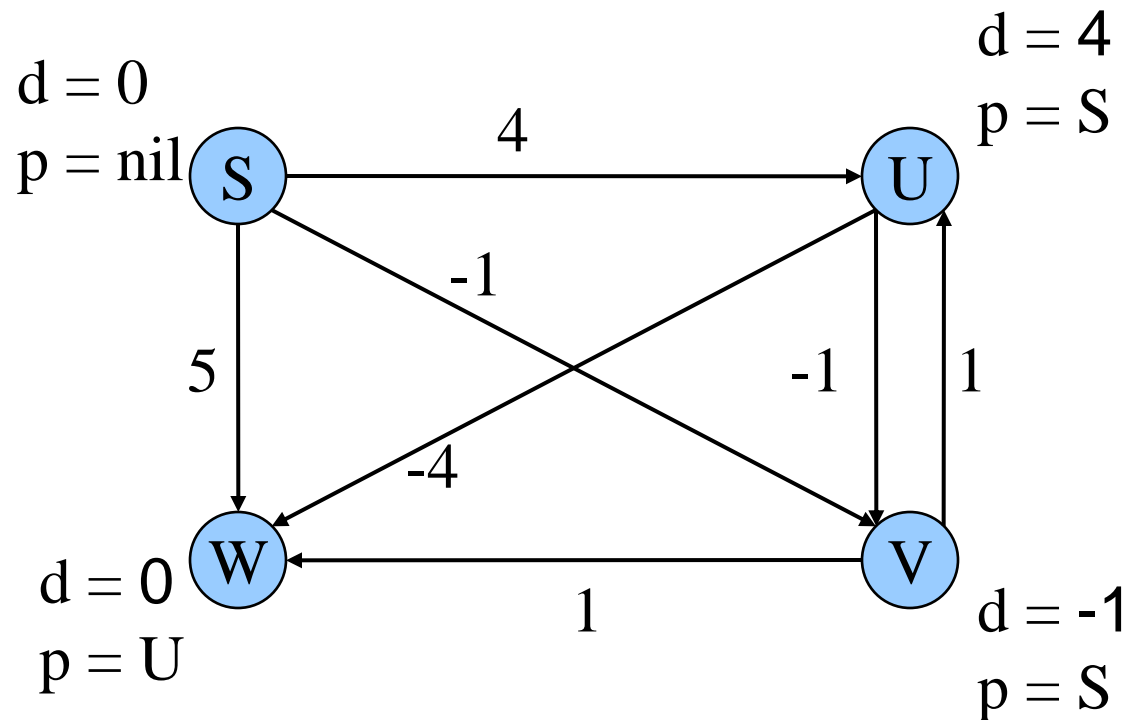


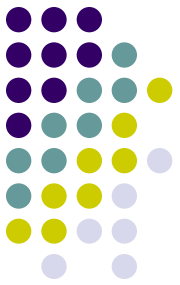


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=1$

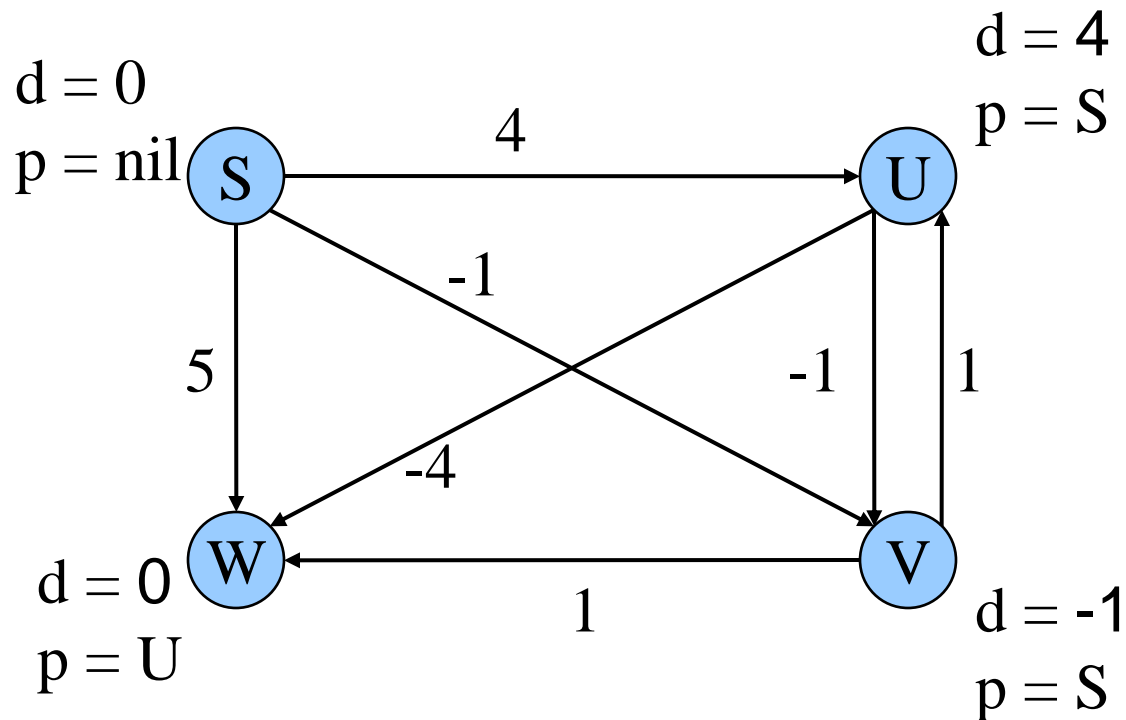


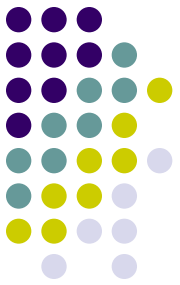


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

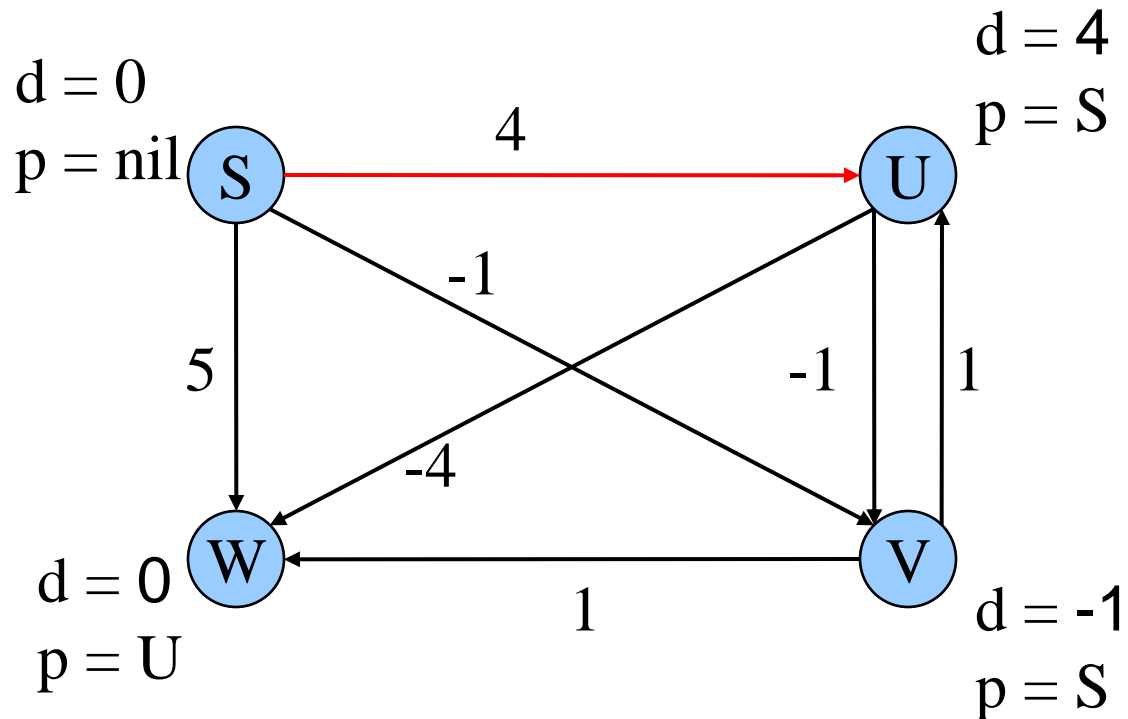


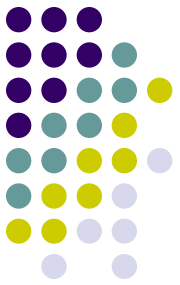


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

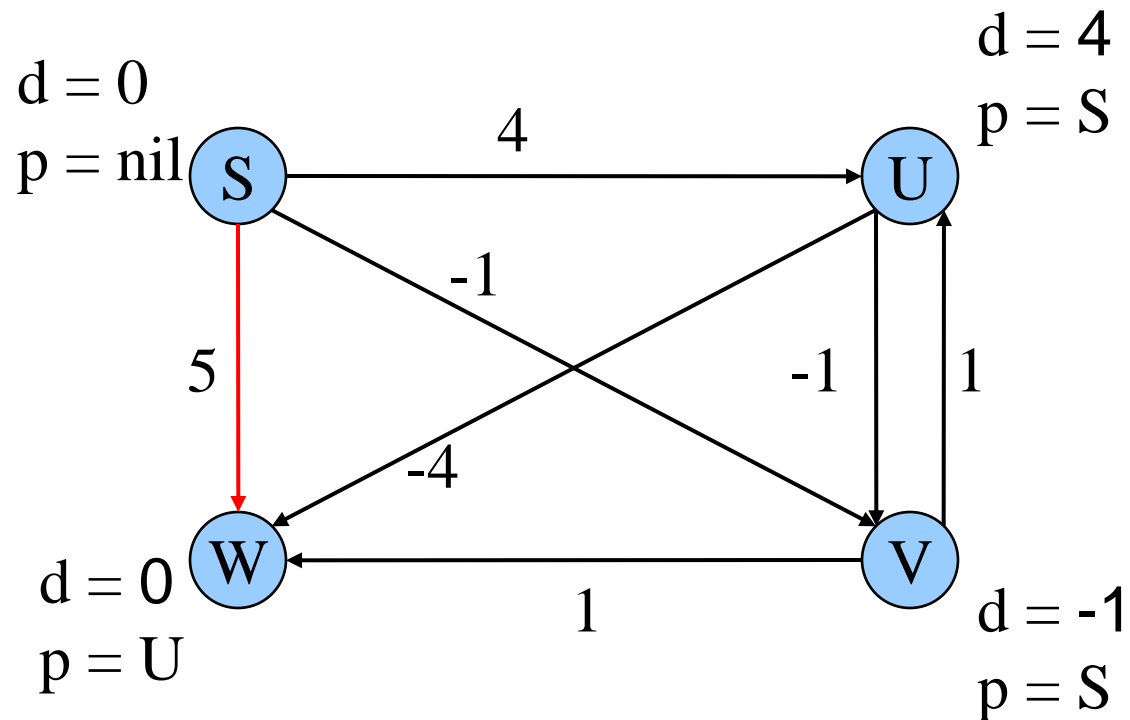




Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

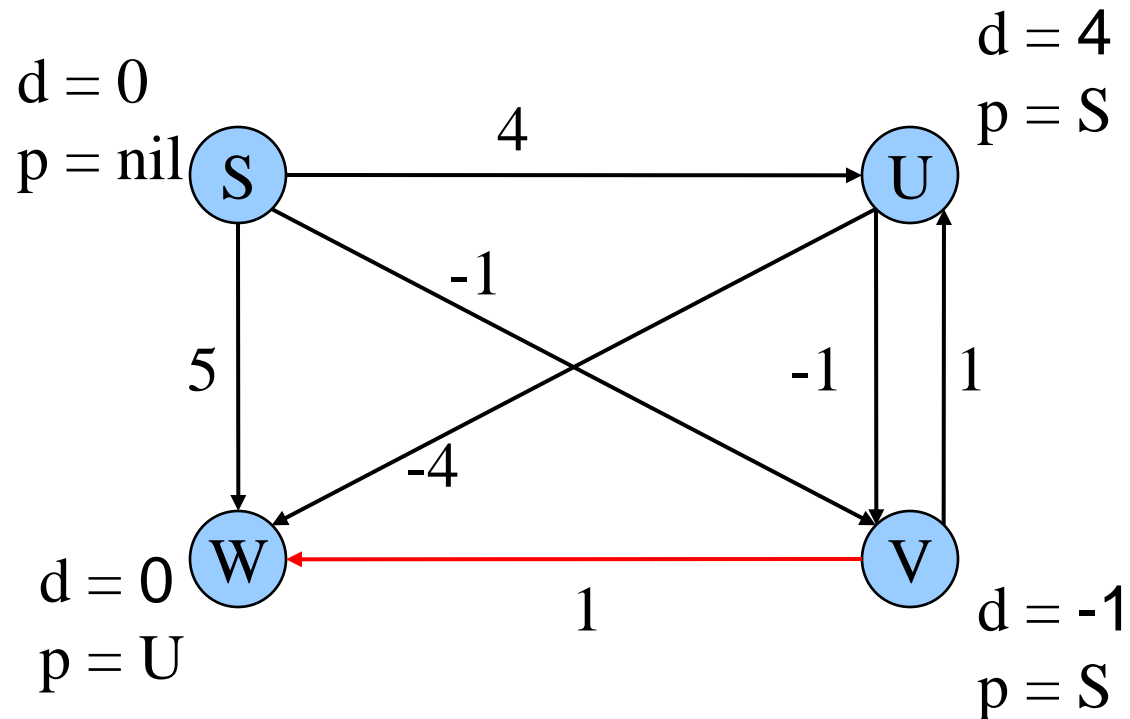


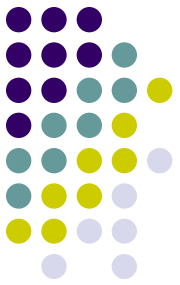


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

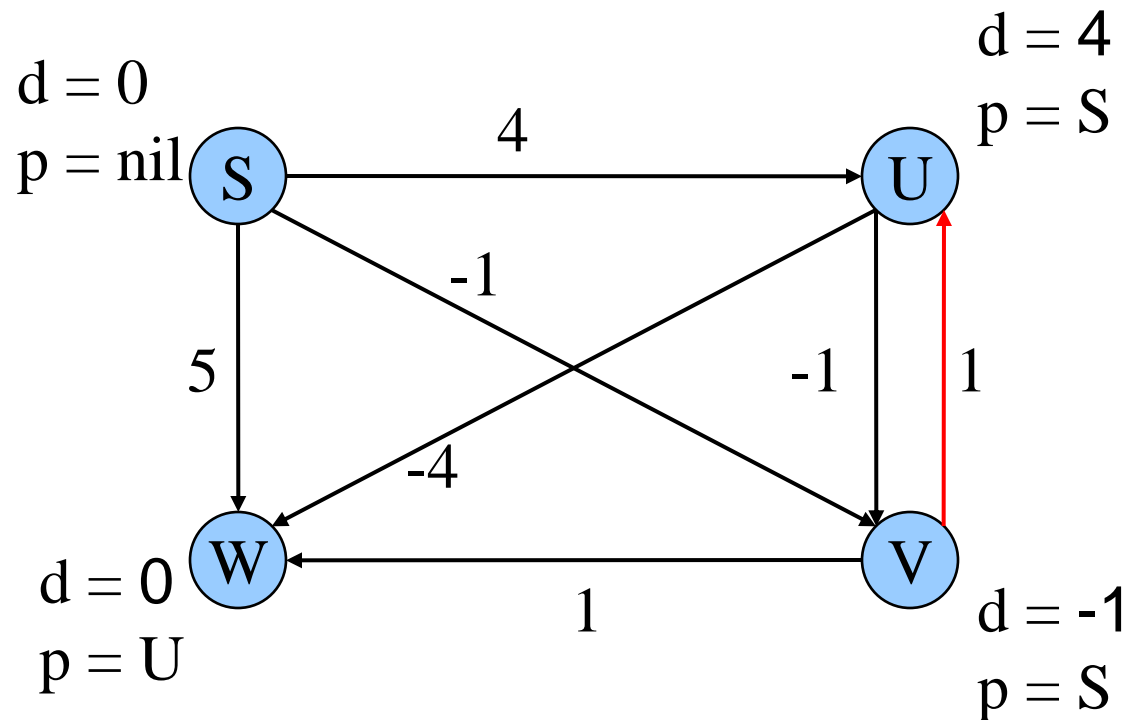


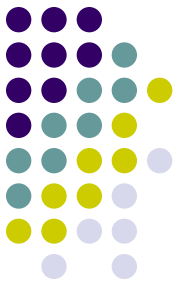


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=2$

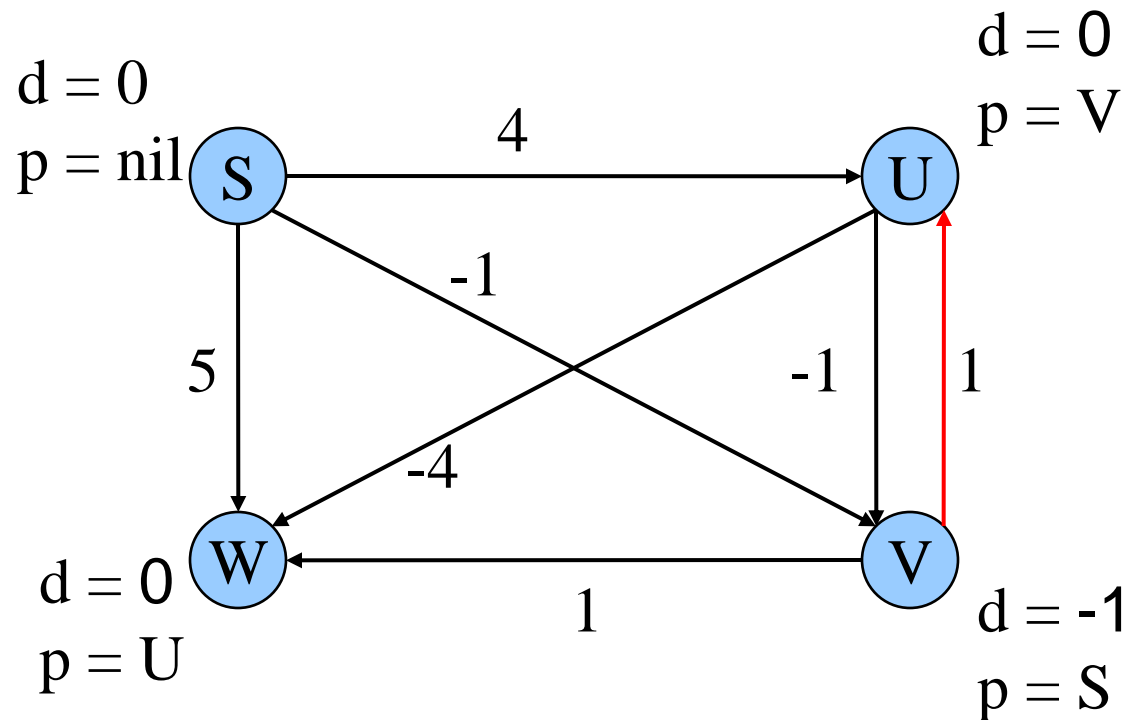


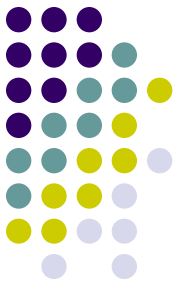


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=2$

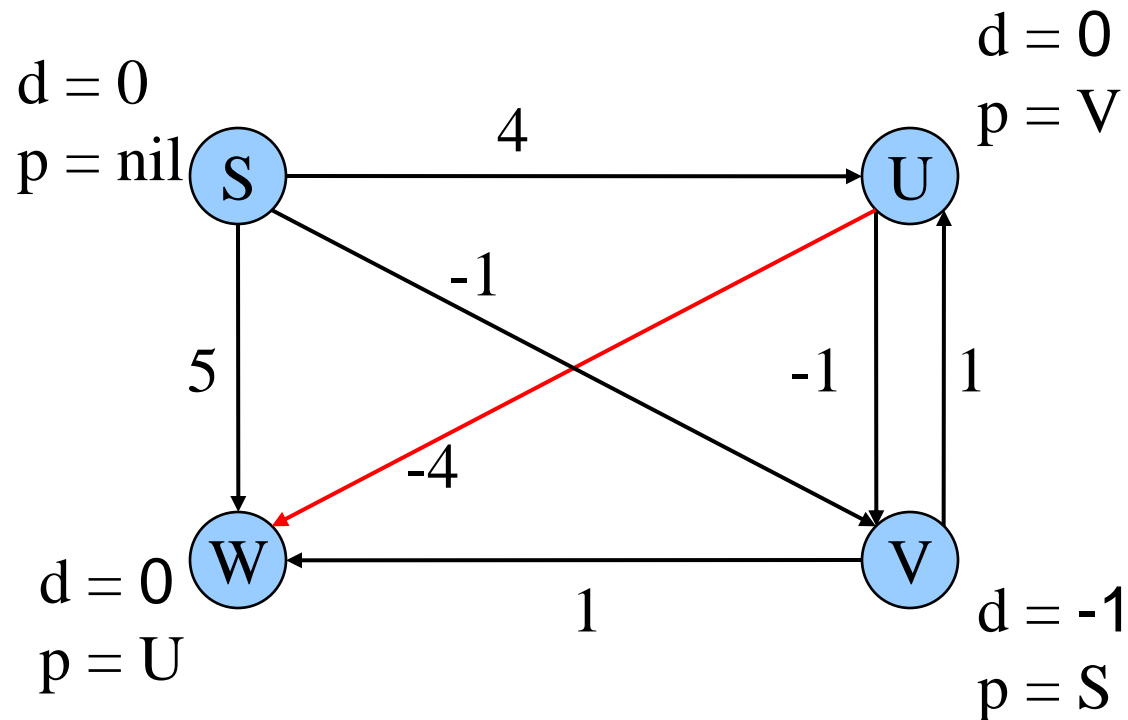




Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

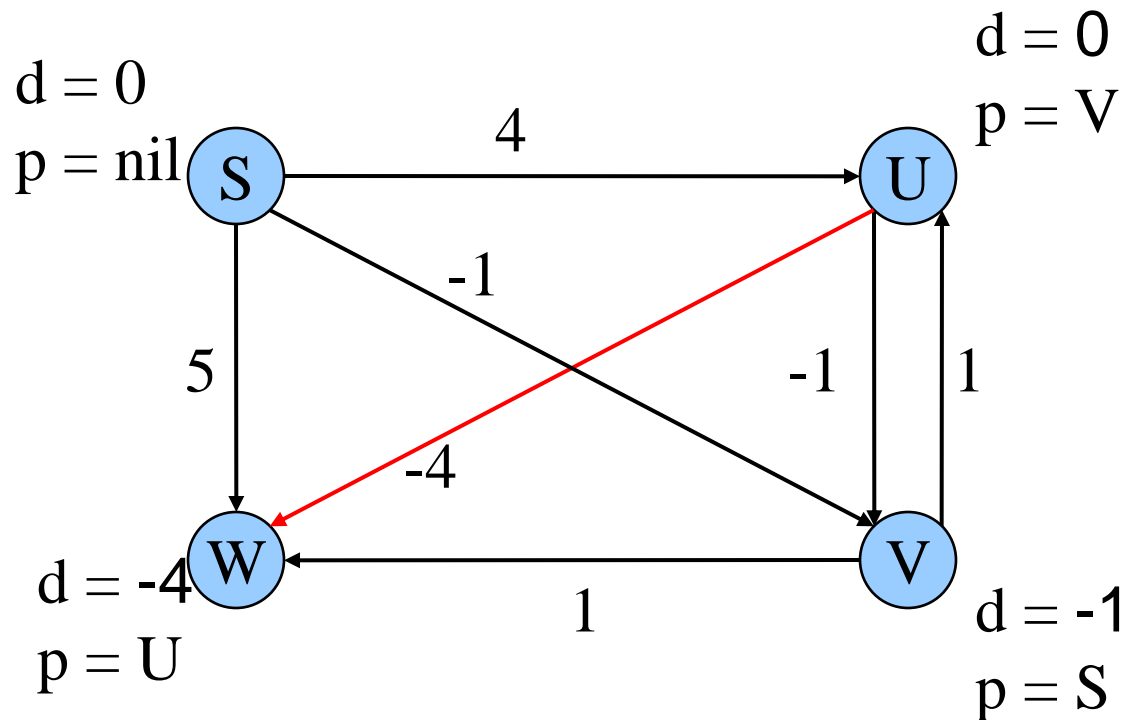


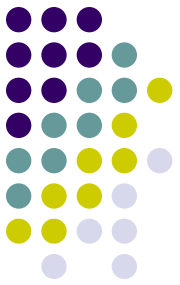


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

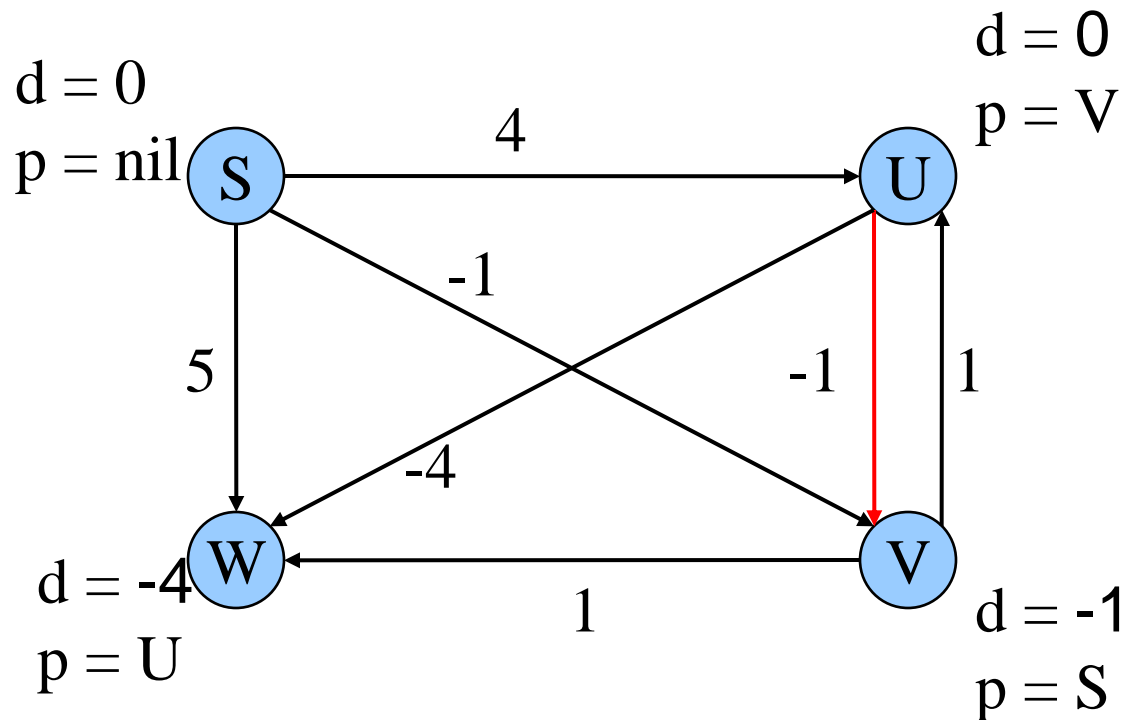


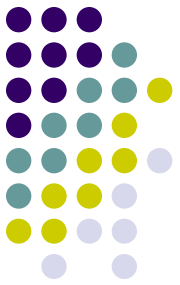


Bellman-Ford

for i in xrange(1, v):
 for (u, v) in edges:
 relax(u, v)

$i=2$

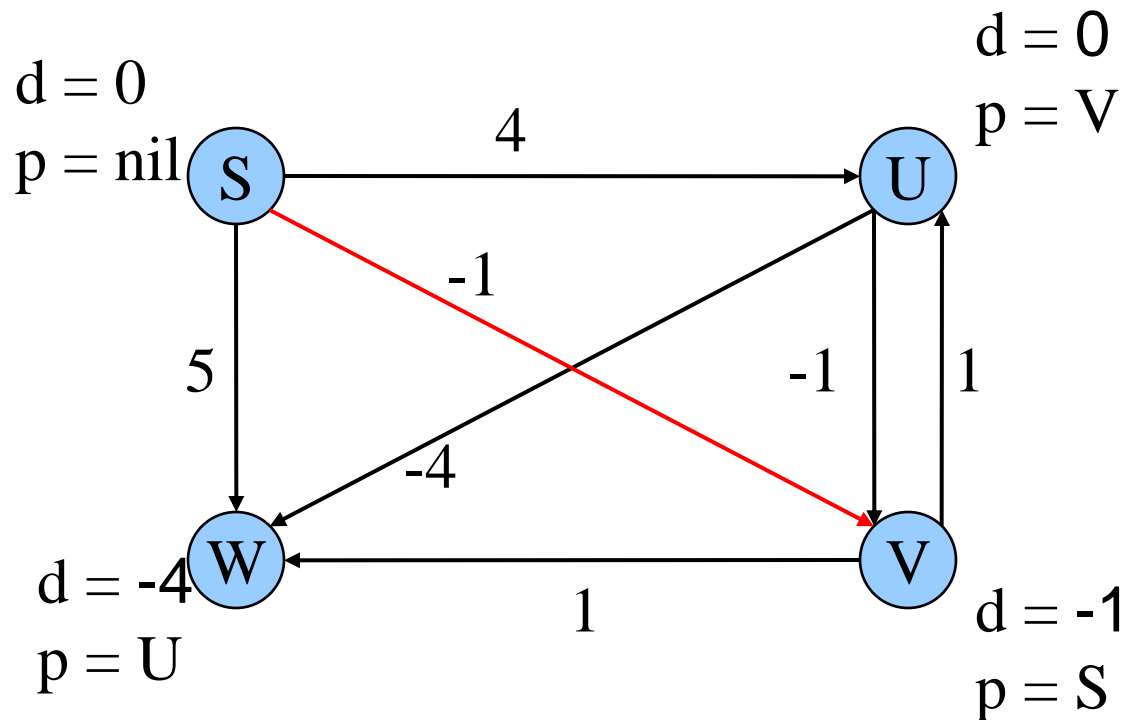


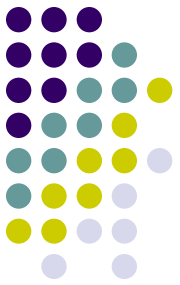


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=2$

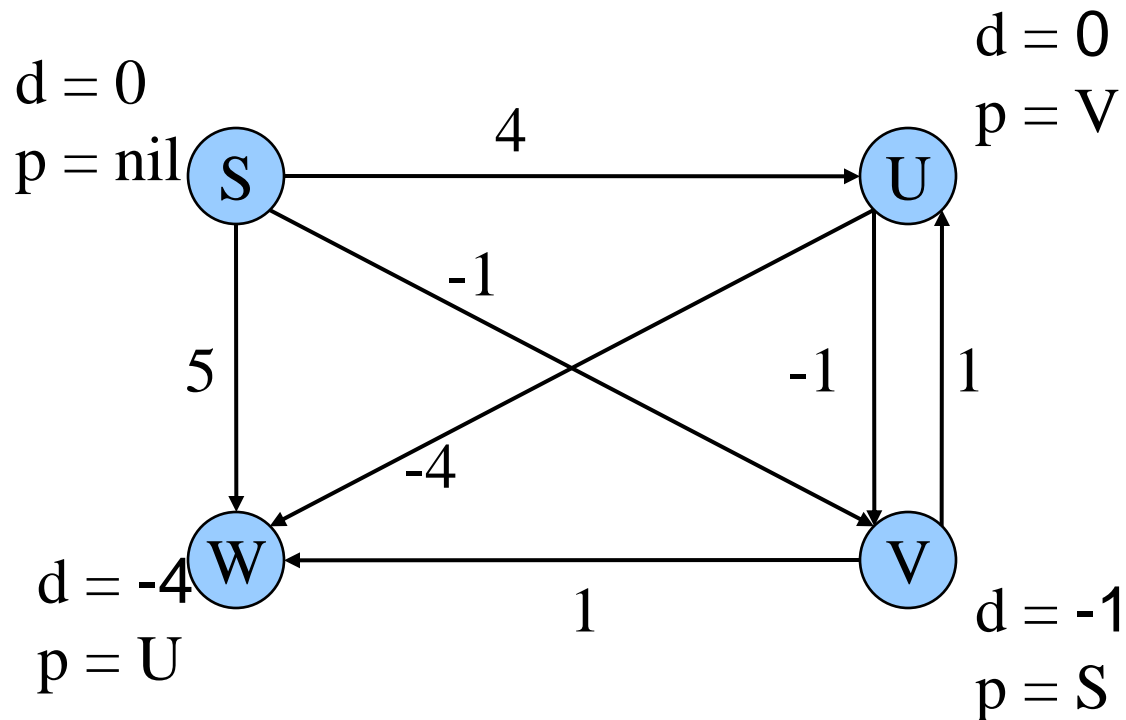


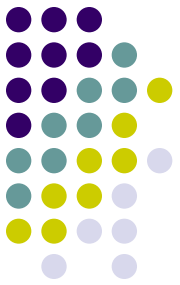


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

$i=2$

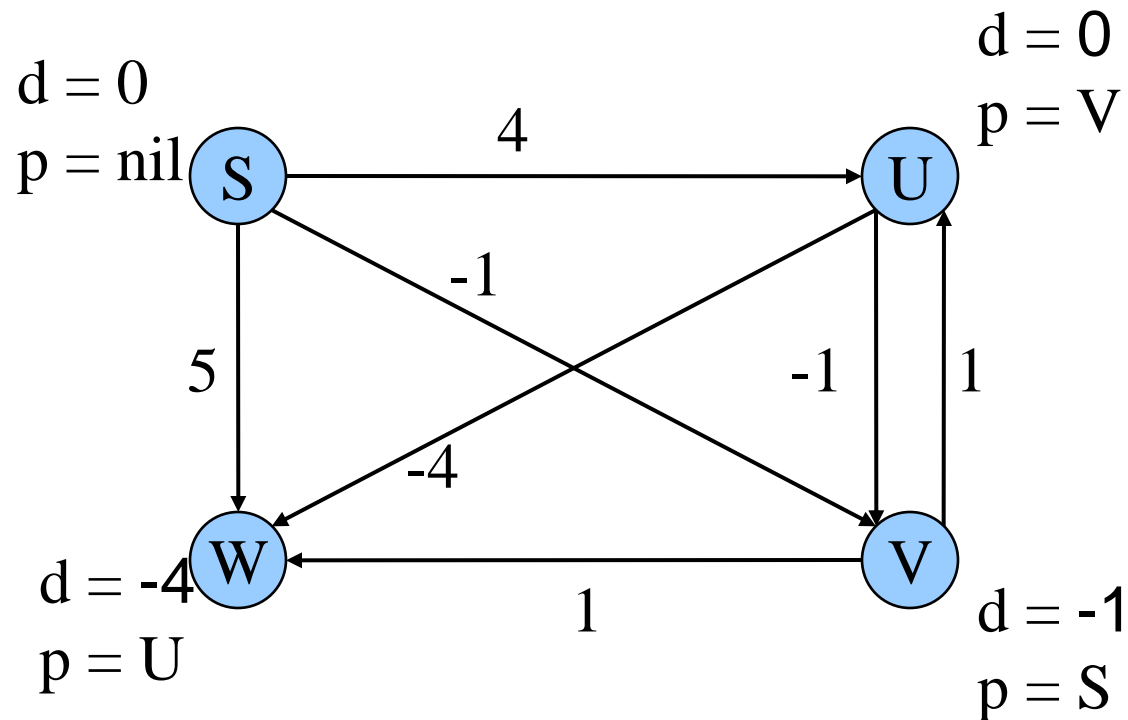


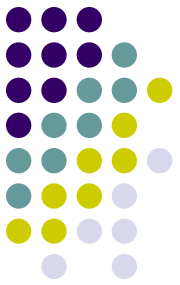


Bellman-Ford

```
for i in xrange(1, v):  
    for (u, v) in edges:  
        relax(u,v)
```

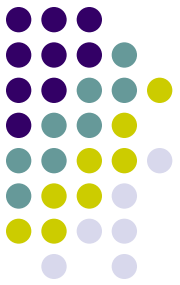
$i=3$





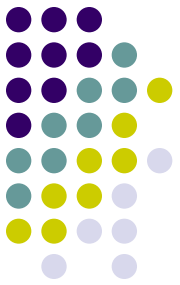
Bellman-Ford

- Hva om vi har en negativ sykel?



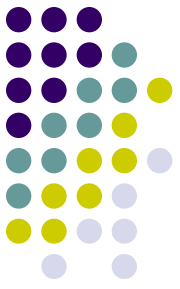
Bellman-Ford

- Hva om vi har en negativ sykel?
- Den vil gi kontinuerlig bedre tilbud



Bellman-Ford

- Hva om vi har en negativ sykel?
- Den vil gi kontinuerlig bedre tilbud
 - Sjekk om vi får forbedringer ved neste iterasjon!

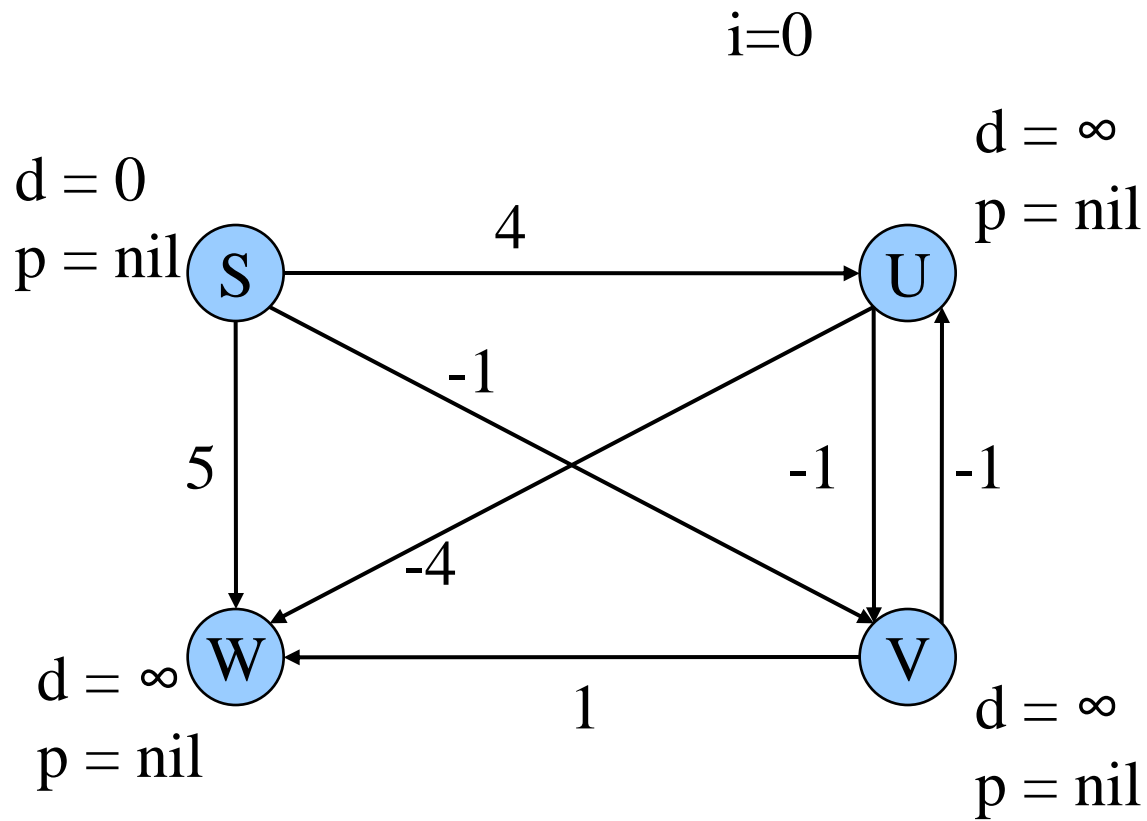


Bellman-Ford

- Hva om vi har en negativ sykel?
- Den vil gi kontinuerlig bedre tilbud
 - Sjekk om vi får forbedringer ved neste iterasjon!
- Minst en node får et kortere estimat ->
 - Vi har en negativ sykel

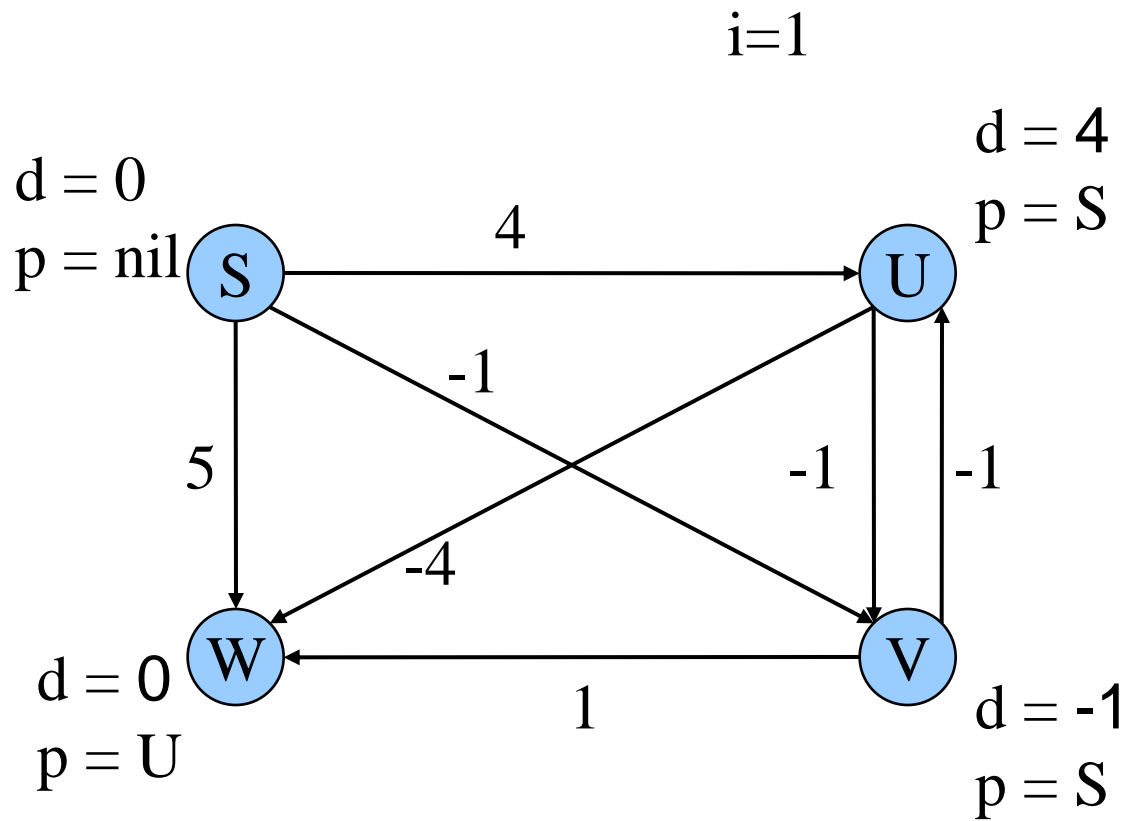


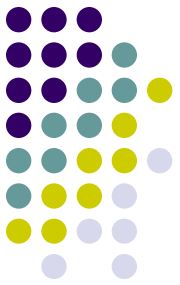
Bellman-Ford



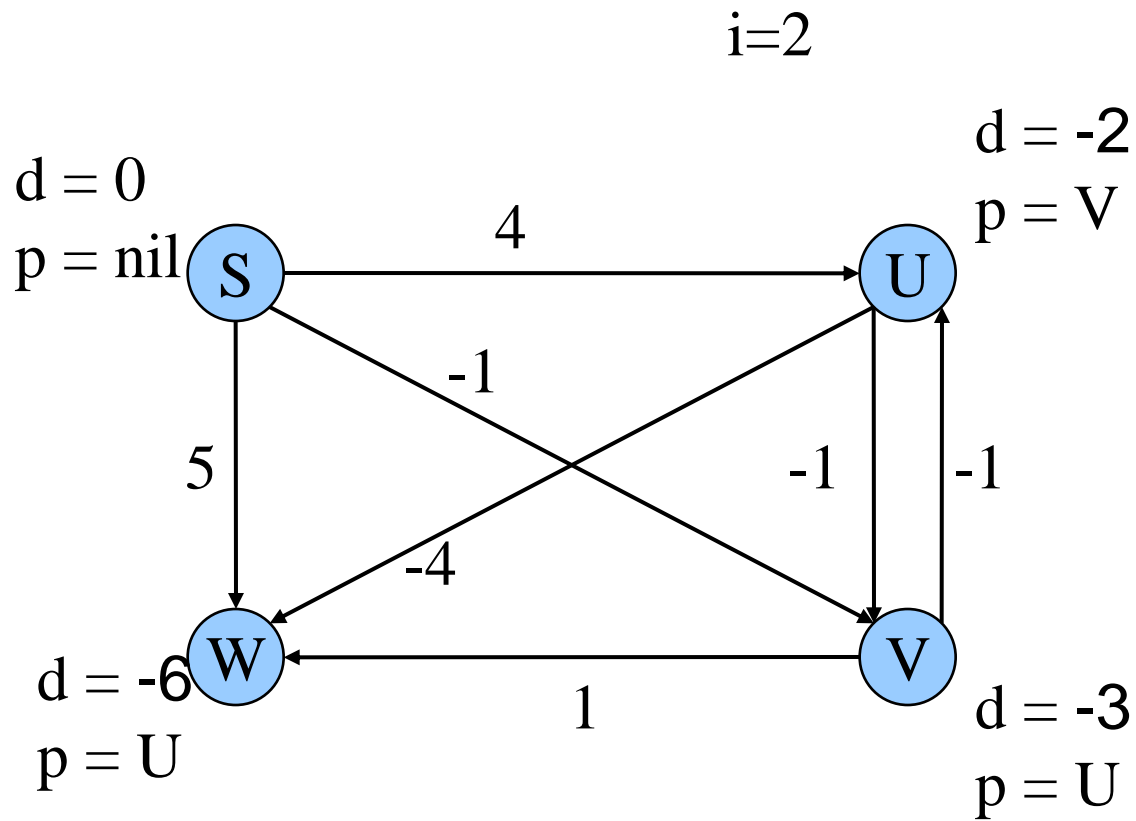


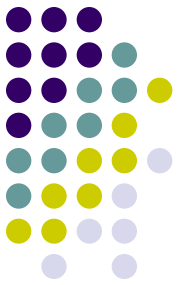
Bellman-Ford



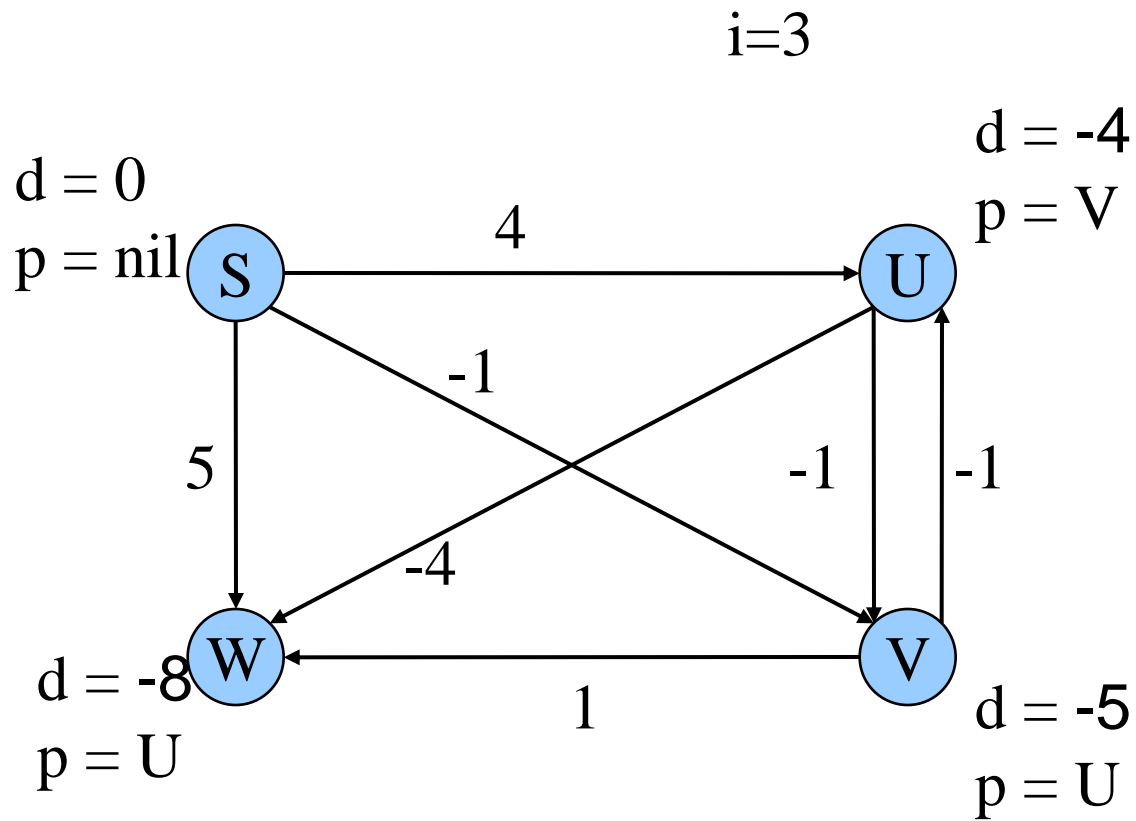


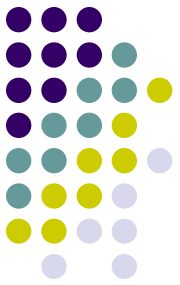
Bellman-Ford



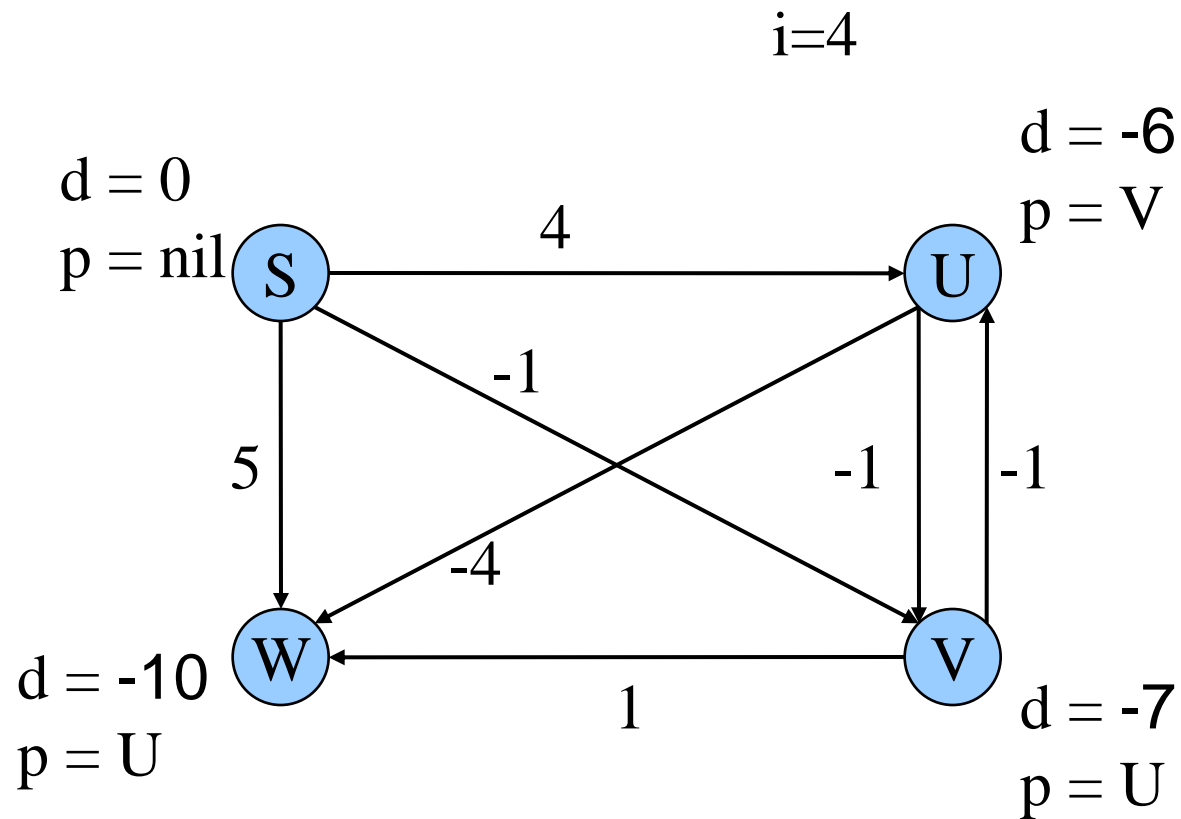


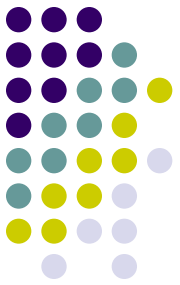
Bellman-Ford





Bellman-Ford





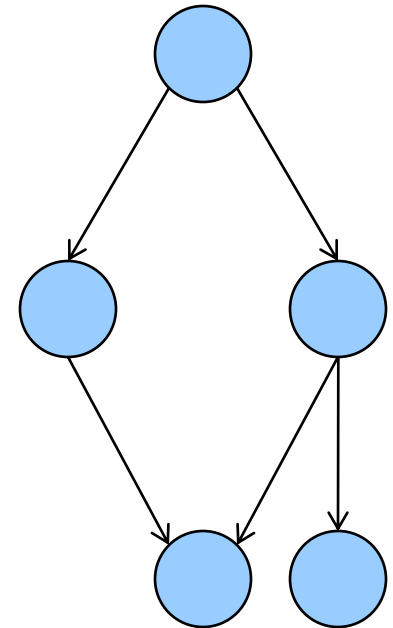
Bellman-Ford

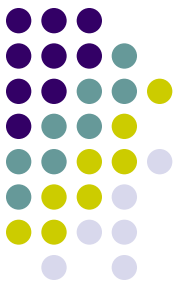
- Kjøretid:
 - Sjekk alle kanter $V-1$ ganger
 - Sjekk etter negativ sykel
 - $O(V \cdot E)$



DAG-shortest-paths

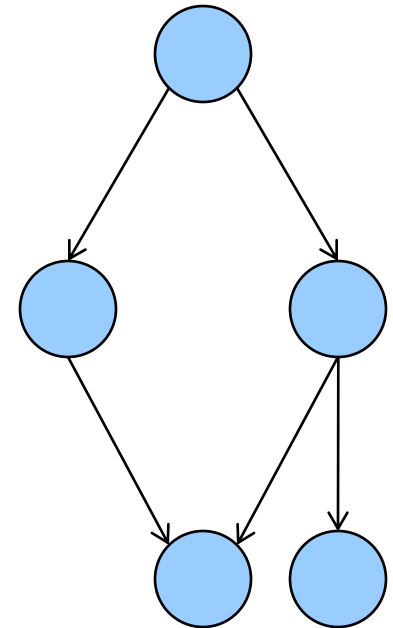
- Hva om grafen er en DAG?

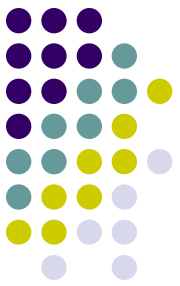




DAG-shortest-paths

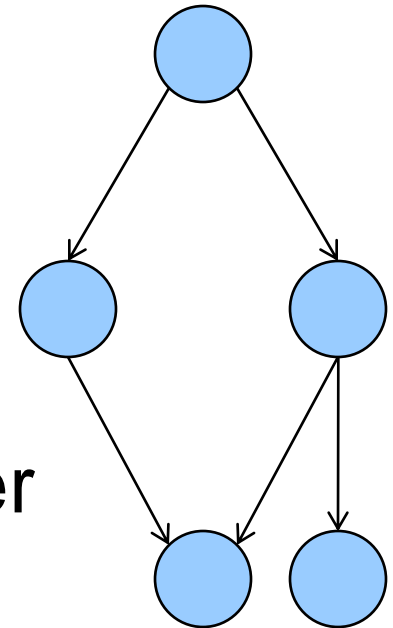
- Hva om grafen er en DAG?
- Kan vi utnytte denne strukturen?

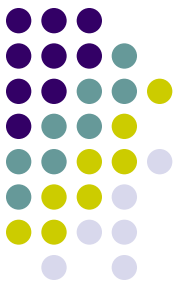




DAG-shortest-paths

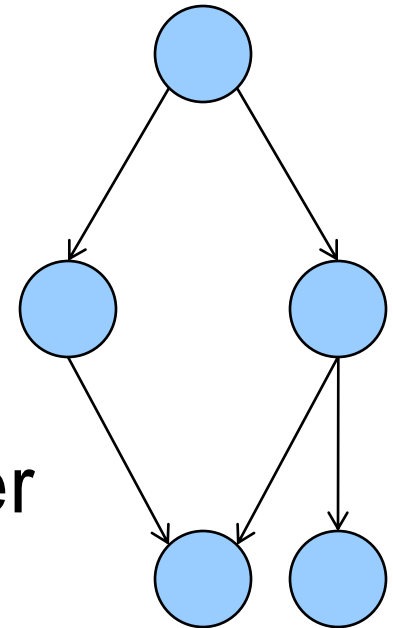
- Hva om grafen er en DAG?
- Kan vi utnytte denne strukturen?
- Ingen sykler -> ingen negative sykler

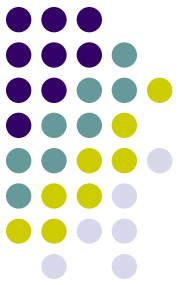




DAG-shortest-paths

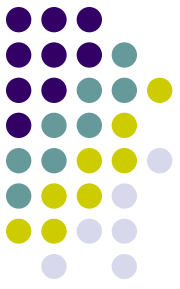
- Hva om grafen er en DAG?
- Kan vi utnytte denne strukturen?
- Ingen sykler -> ingen negative sykler
- Vi kan lage en topologisk sortering





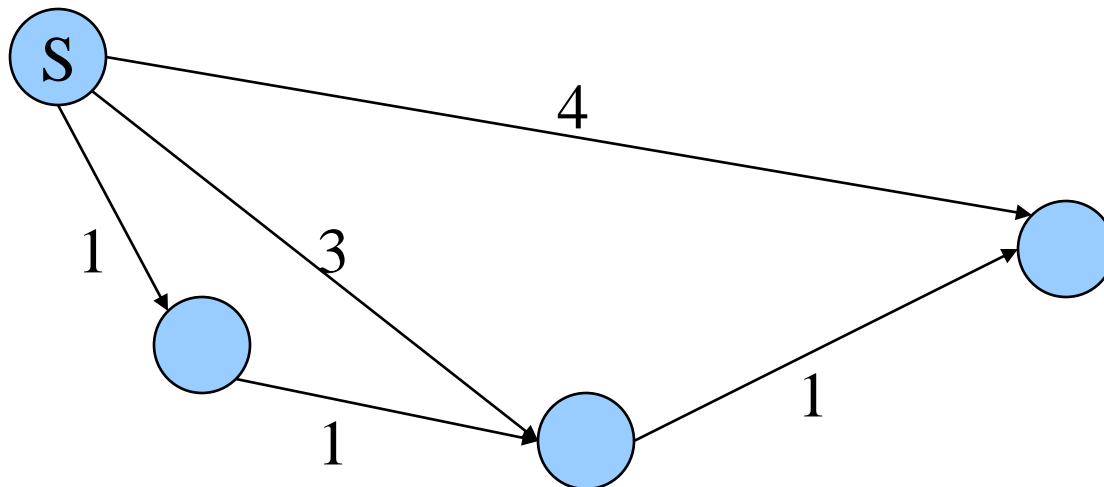
DAG-shortest-paths

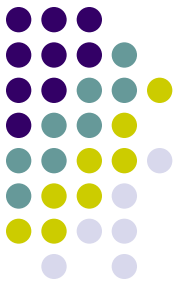
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:



DAG-shortest-paths

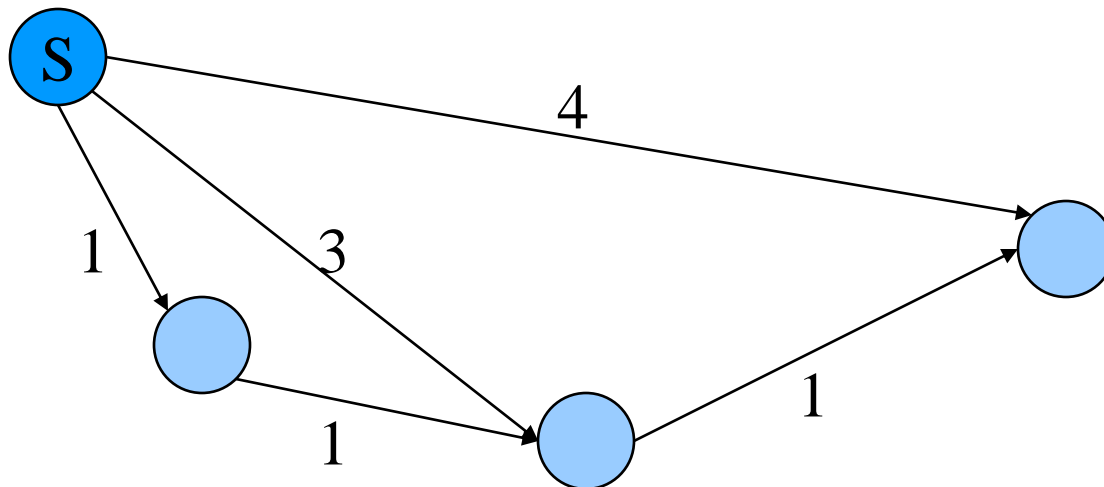
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:

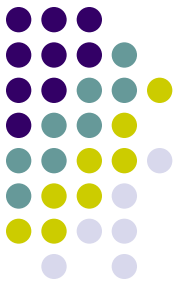




DAG-shortest-paths

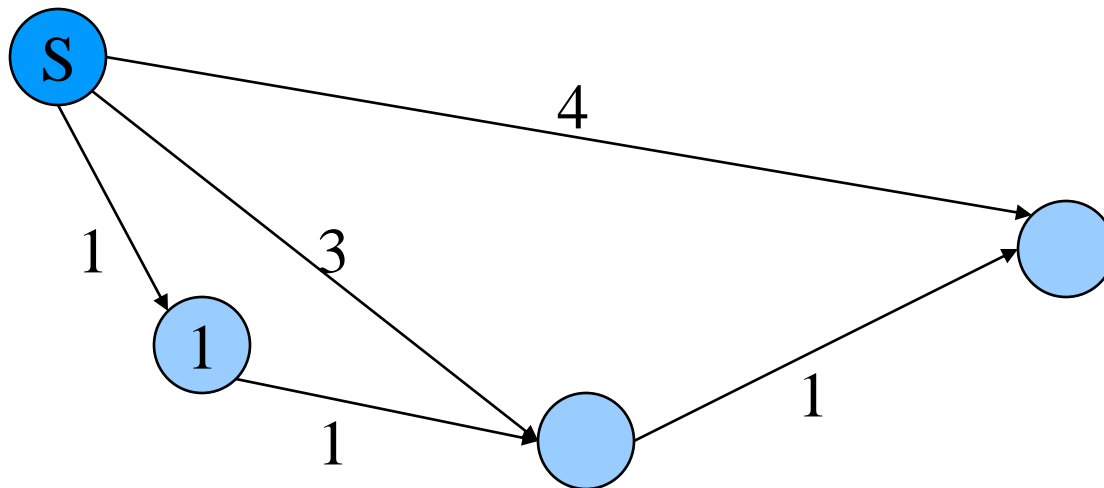
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:





DAG-shortest-paths

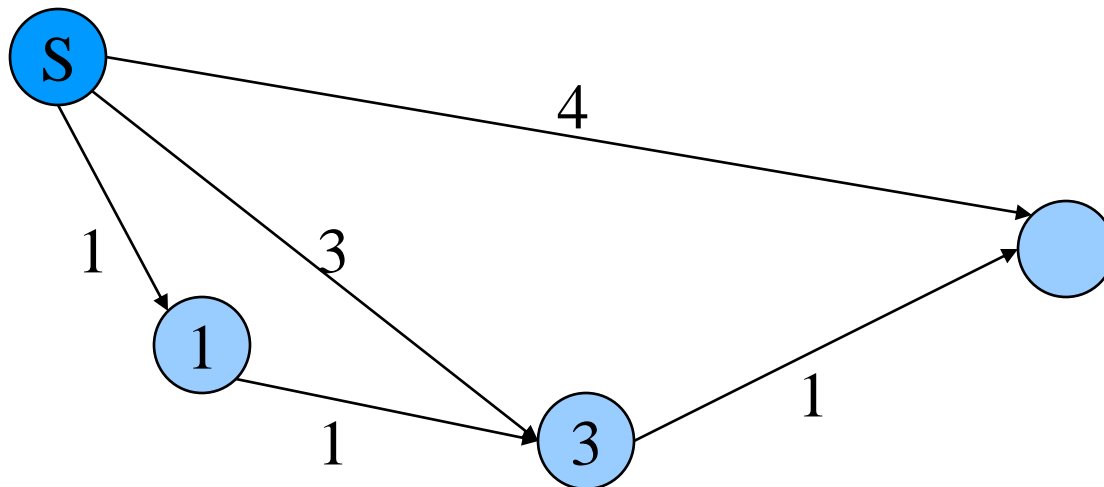
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:

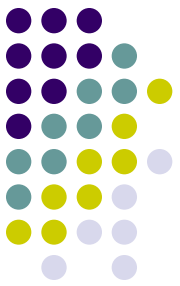




DAG-shortest-paths

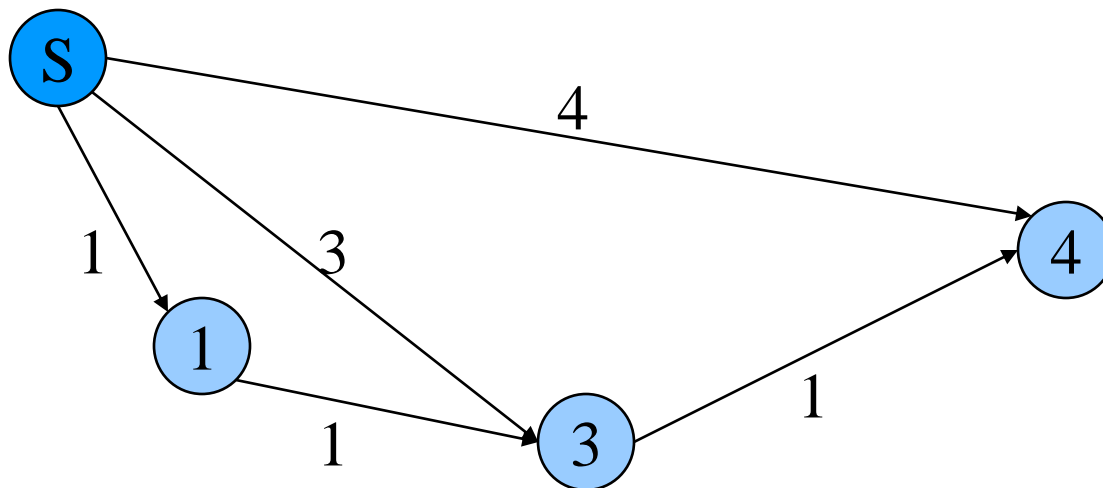
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:





DAG-shortest-paths

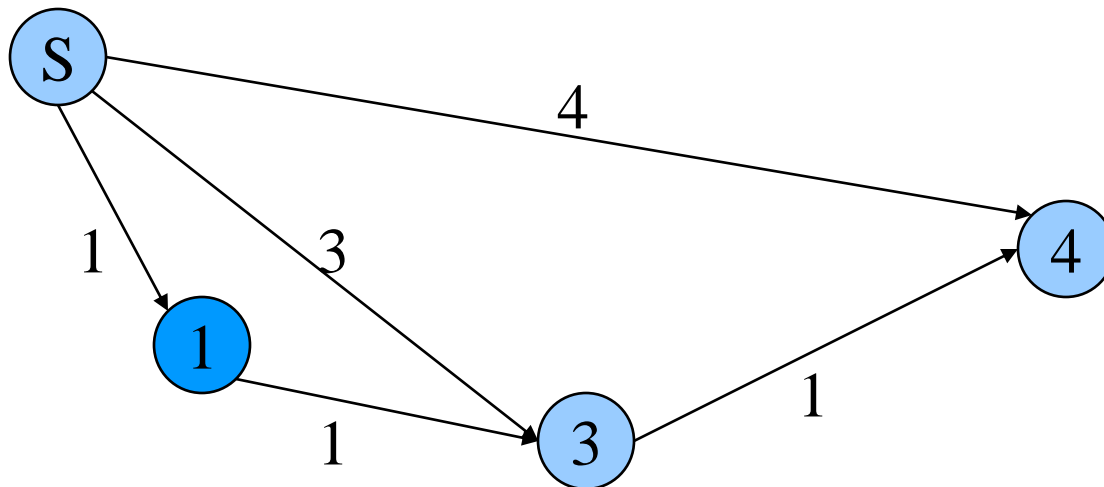
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:





DAG-shortest-paths

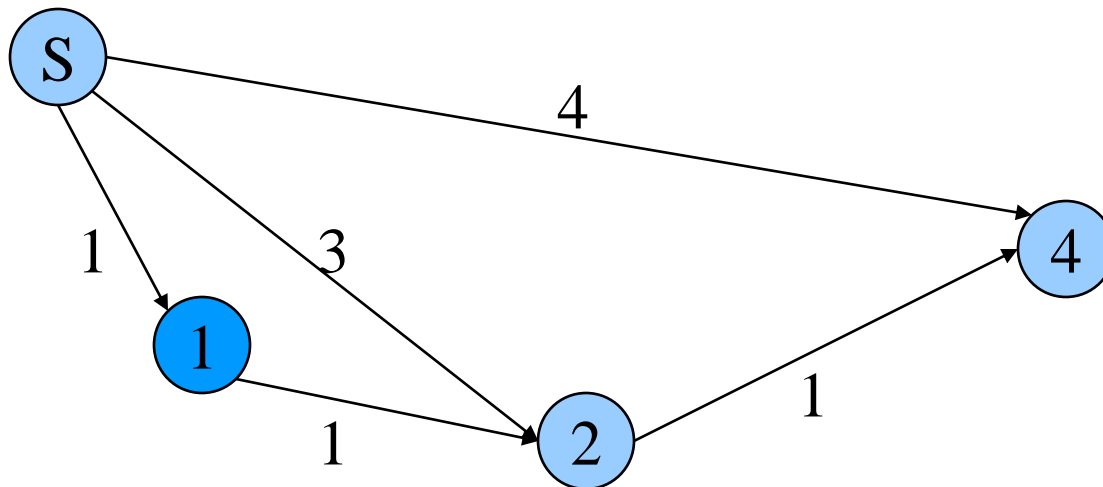
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:

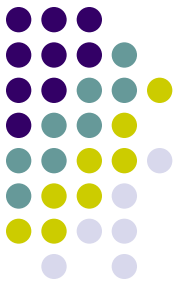




DAG-shortest-paths

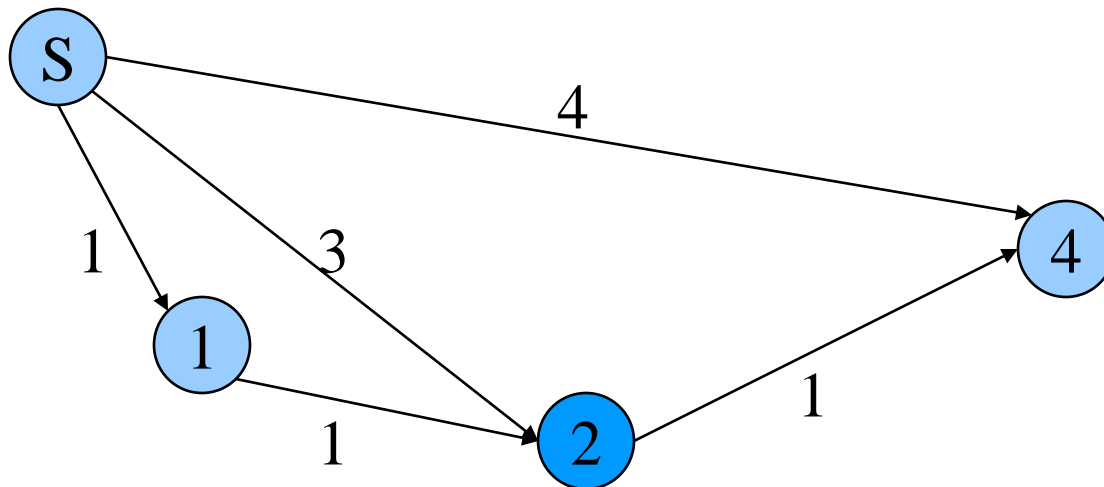
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:

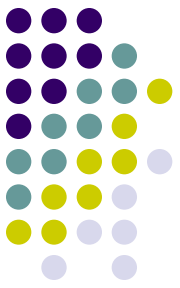




DAG-shortest-paths

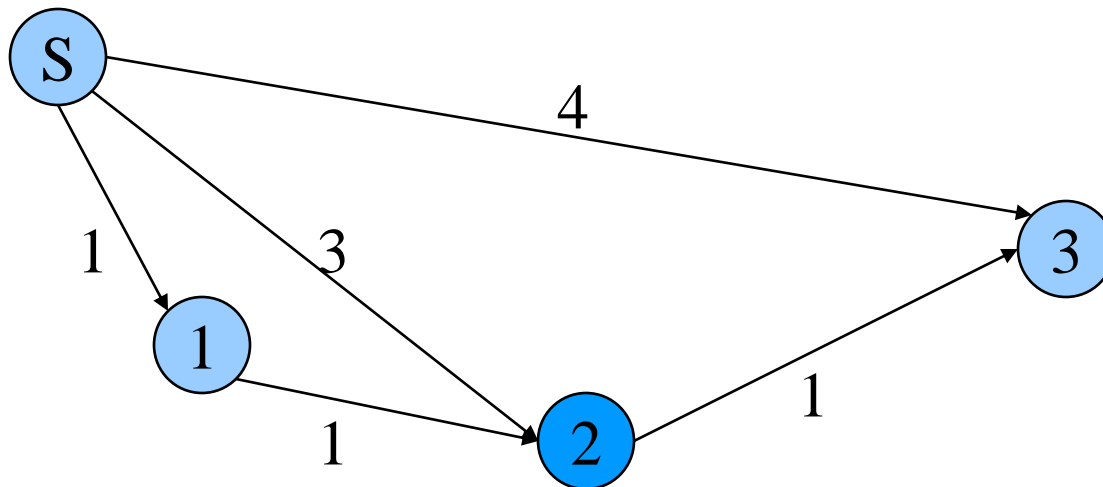
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:





DAG-shortest-paths

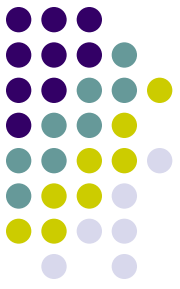
- Topologisk sortering:
 - Lar oss besøke i en slik rekkefølge at estimatene ikke kan oppdateres etter at en node har blitt besøkt:





DAG-shortest-paths

- Kjøretid:
 - Går igjennom alle nodene og alle kantene
 - $O(V + E)$



Korteste vei - alle-til-alle

- Resultatet blir en $V \times V$ matrise der element (A, B) representerer minste avstand fra A til B



Korteste vei - alle-til-alle

- Resultatet blir en $V \times V$ matrise der element (A, B) representerer minste avstand fra A til B
- Vi kan kjøre Dijkstra eller Bellman-Ford V ganger
 - Dijkstra: $O(V^3)$ eller $O(V E \lg V)$
 - Bellman-Ford: $O(V^2 E)$



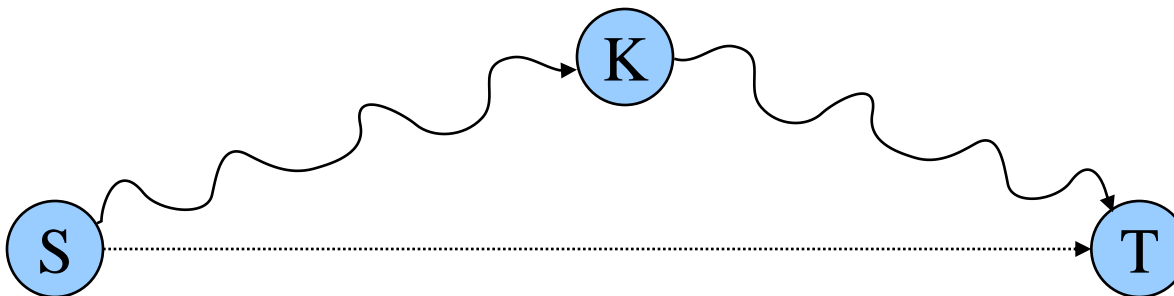
Korteste vei - alle-til-alle

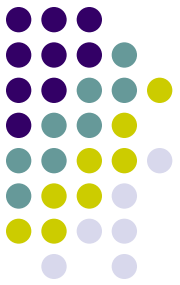
- Resultatet blir en $V \times V$ matrise der element (A, B) representerer minste avstand fra A til B
- Vi kan kjøre Dijkstra eller Bellman-Ford V ganger
 - Dijkstra: $O(V^3)$ eller $O(V E \lg V)$
 - Bellman-Ford: $O(V^2 E)$
- Finnes spesialiserte løsninger



Korteste vei - alle-til-alle

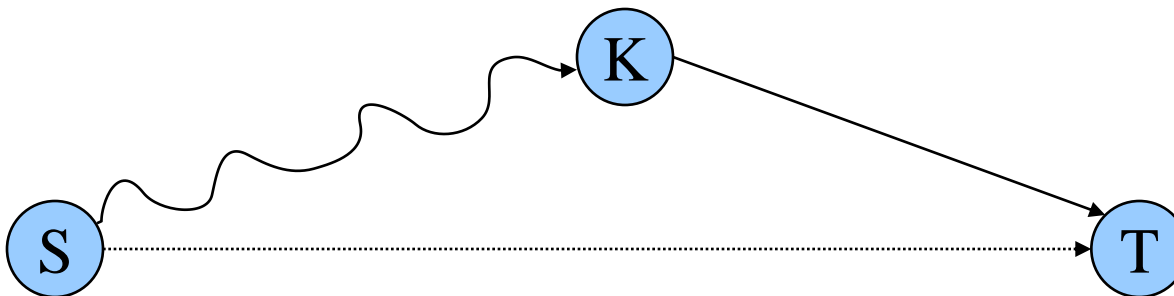
- Vi kan igjen utnytte at en dekomponert korteste vei består av korteste veier:

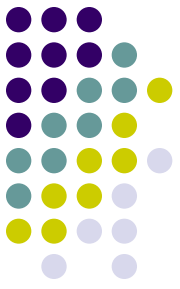




Korteste vei - alle-til-alle

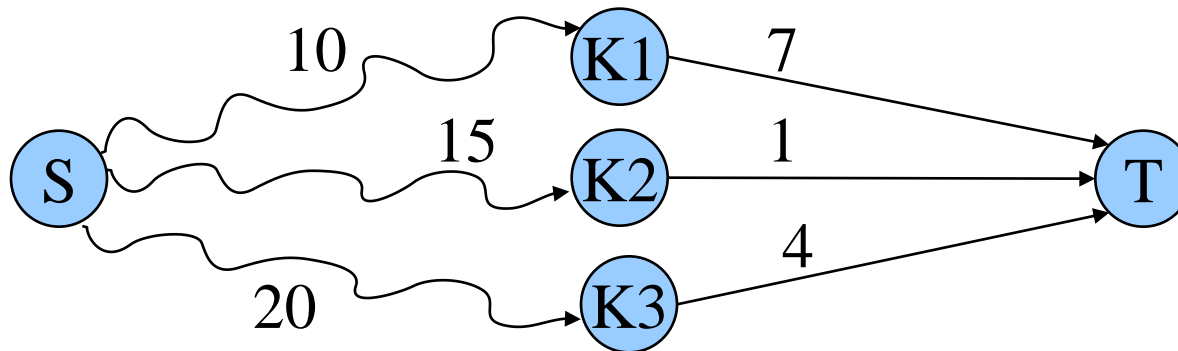
- Spesialtilfelle:
 - Den korteste veien fra S til T kan bygges opp med den korteste fra S til K, og kanten mellom K og T
 - K representerer her den siste noden før T

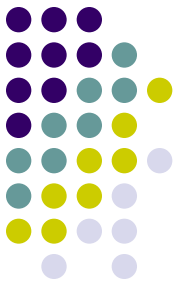




Korteste vei - alle-til-alle

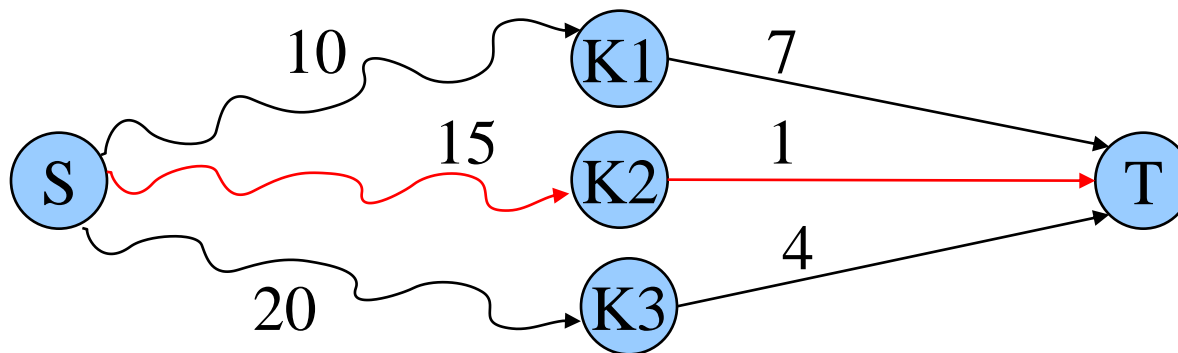
- Korteste vei av lengde i kan settes sammen av en korteste avstand av lengde $i-1$ + en kant
 - Må sjekke alle muligheter

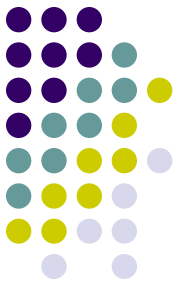




Korteste vei - alle-til-alle

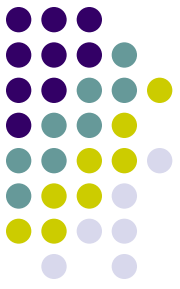
- Korteste vei av lengde i kan settes sammen av en korteste avstand av lengde $i-1$ + en kant
 - Må sjekke alle muligheter





Korteste vei - alle-til-alle

- Matematisk: $l_{ij}^{(m)}$ = *korteste vei ved bruk av maks m kanter*



Korteste vei - alle-til-alle

- Matematisk: $l_{ij}^{(m)}$ = korteste vei ved bruk av maks m kanter

$$l_{ij}^{(0)} = \begin{cases} 0, & \text{hvis } i = j \\ \infty, & \text{hvis } i \neq j \end{cases}$$



Korteste vei - alle-til-alle

- Matematisk: $l_{ij}^{(m)}$ = korteste vei ved bruk av maks m kanter

$$l_{ij}^{(0)} = \begin{cases} 0, & \text{hvis } i = j \\ \infty, & \text{hvis } i \neq j \end{cases}$$

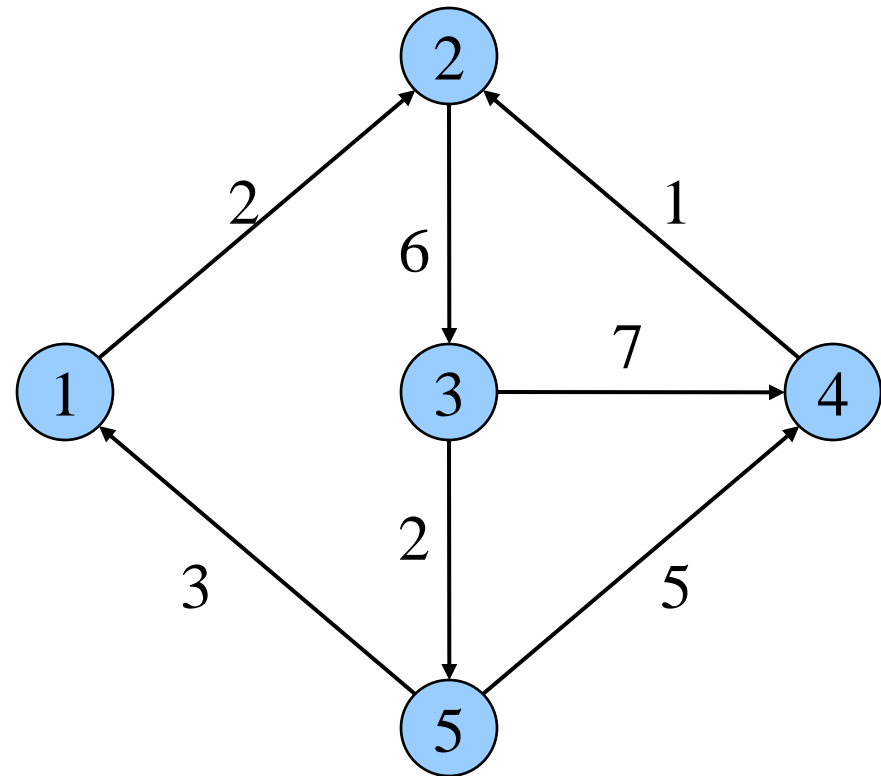
$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\}$$



Korteste vei - alle-til-alle

- Eksempel:

$$l^{(1)} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 0 \\ - \\ - \\ - \\ 3 \end{array} \begin{array}{c} 2 \\ 0 \\ - \\ 1 \\ - \end{array} \begin{array}{c} - \\ 6 \\ 0 \\ - \\ - \end{array} \begin{array}{c} - \\ - \\ 7 \\ 0 \\ 5 \end{array} \begin{array}{c} - \\ - \\ 2 \\ - \\ 0 \end{array}$$

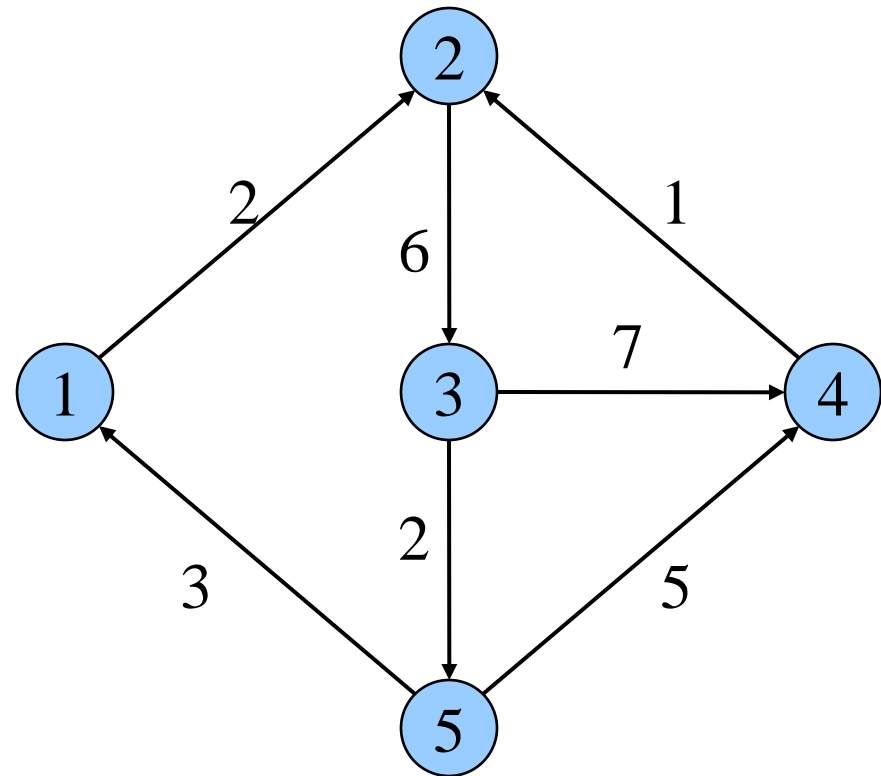


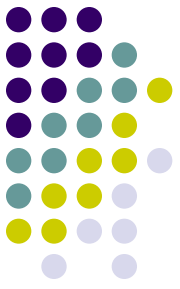


Korteste vei - alle-til-alle

- Eksempel:

$$W = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 2 & - & - & - \\ - & 0 & 6 & - & - \\ - & - & 0 & 7 & 2 \\ - & 1 & - & 0 & - \\ 3 & - & - & 5 & 0 \end{bmatrix} \end{matrix}$$

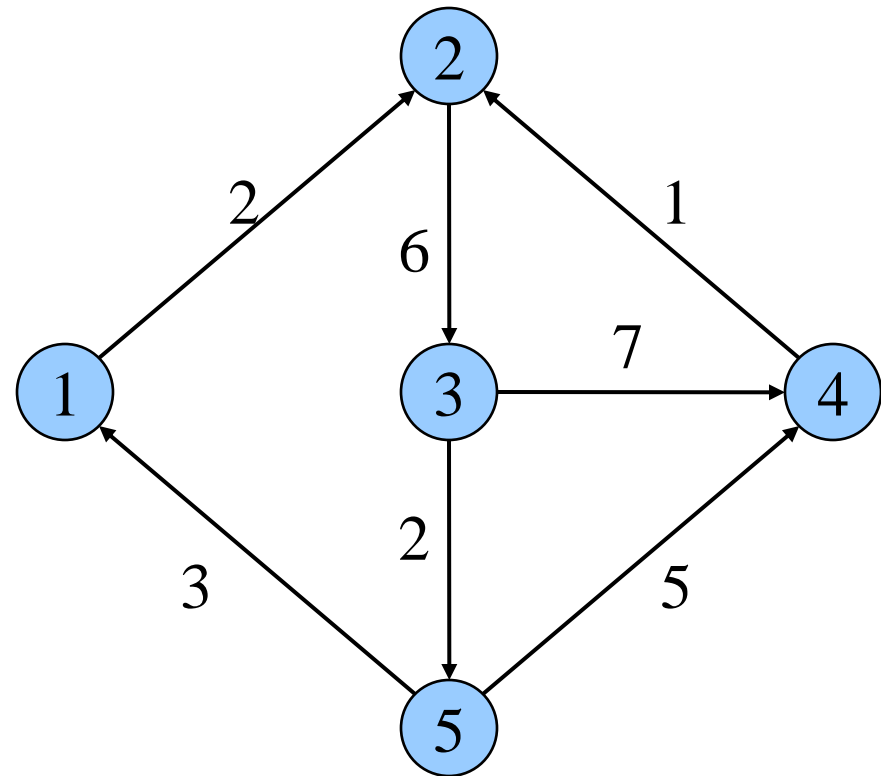


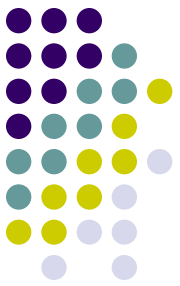


Korteste vei - alle-til-alle

- Eksempel:

$$l^{(1)} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 0 \\ - \\ - \\ - \\ 3 \end{array} \begin{array}{c} 2 \\ 0 \\ - \\ 1 \\ - \end{array} \begin{array}{c} - \\ 6 \\ 0 \\ - \\ - \end{array} \begin{array}{c} - \\ - \\ 7 \\ 0 \\ 5 \end{array} \begin{array}{c} - \\ - \\ 2 \\ - \\ 0 \end{array}$$

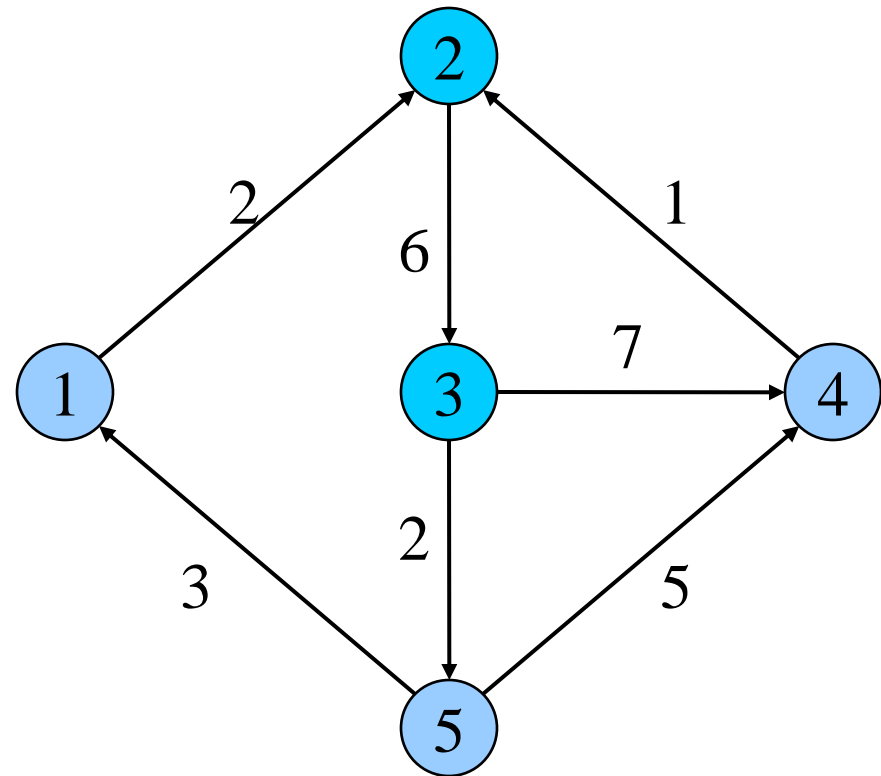


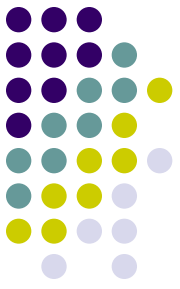


Korteste vei - alle-til-alle

- Eksempel:

$$l^{(3)} = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 2 & 8 & 15 & 10 \\ 11 & 0 & 6 & 13 & 8 \\ 5 & 8 & 0 & 7 & 2 \\ - & 1 & 7 & 0 & - \\ 3 & 5 & - & 5 & 0 \end{bmatrix} \end{matrix} \quad k=$$

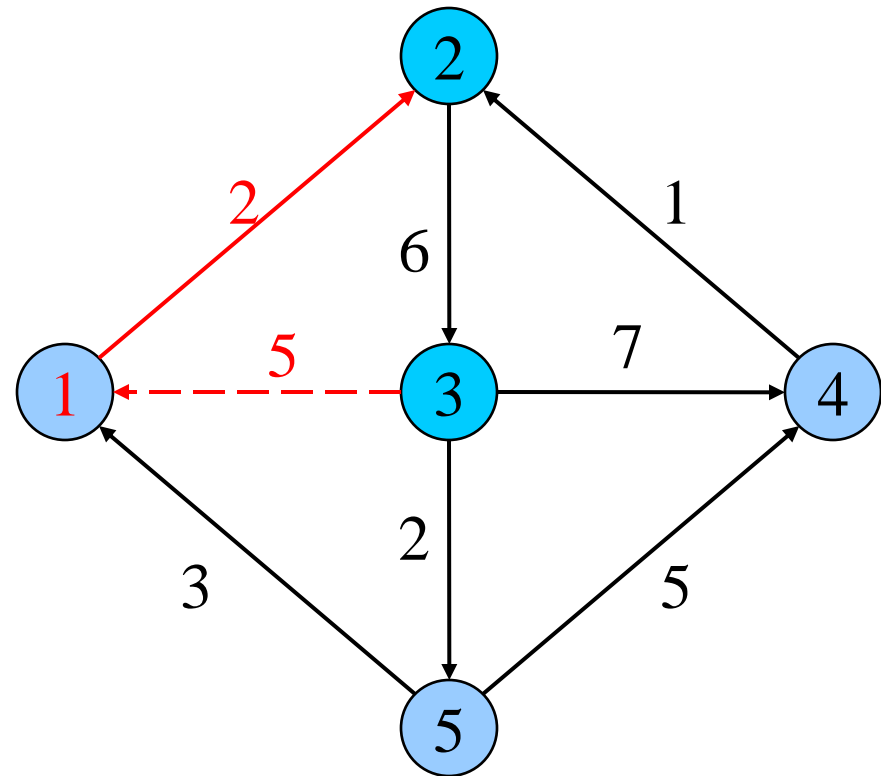


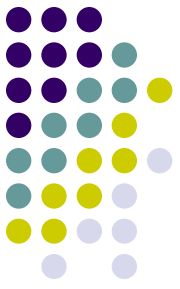


Korteste vei - alle-til-alle

- Eksempel:

k=1

$$l^{(3)} = \begin{matrix} & \mathbf{1} & \mathbf{2} & 3 & 4 & 5 \\ \begin{matrix} 1 \\ 2 \\ \mathbf{3} \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 2 & 8 & 15 & 10 \\ 11 & 0 & 6 & 13 & 8 \\ \mathbf{5} & \mathbf{8} & 0 & 7 & 2 \\ - & 1 & 7 & 0 & - \\ 3 & 5 & - & 5 & 0 \end{bmatrix} \end{matrix}$$




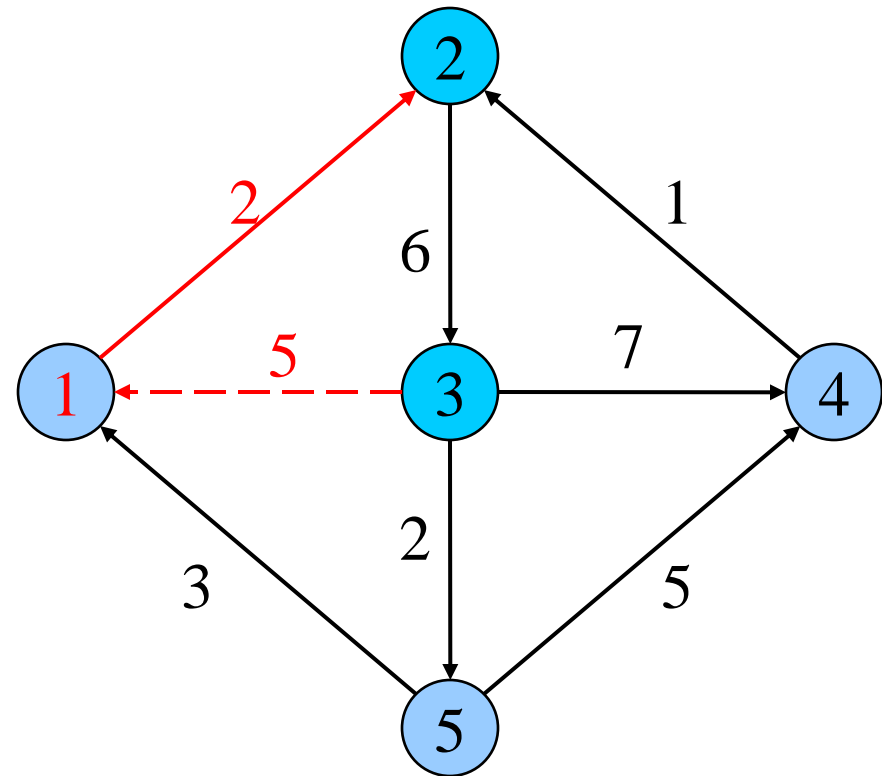
Korteste vei - alle-til-alle

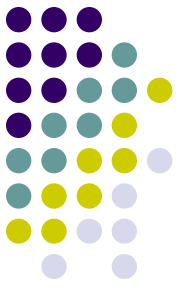
- Eksempel:

k=1

$l^{(3)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	11	0	6	13	8
3	5	7	0	7	2
4	-	1	7	0	-
5	3	5	-	5	0





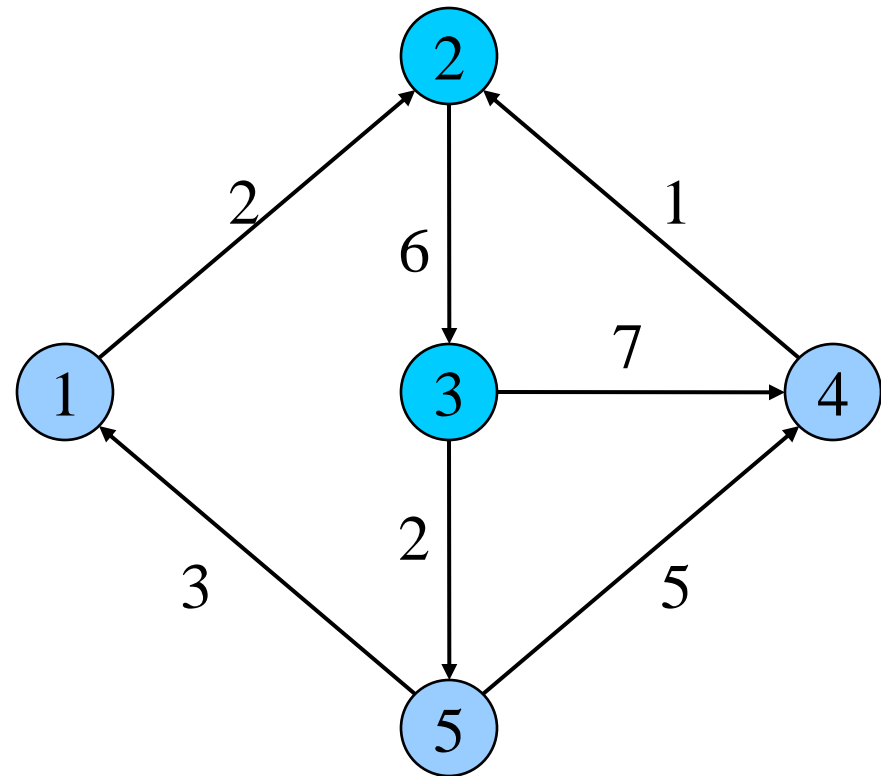
Korteste vei - alle-til-alle

- Eksempel:

k=...

$l^{(3)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	11	0	6	13	8
3	5	7	0	7	2
4	-	1	7	0	-
5	3	5	-	5	0

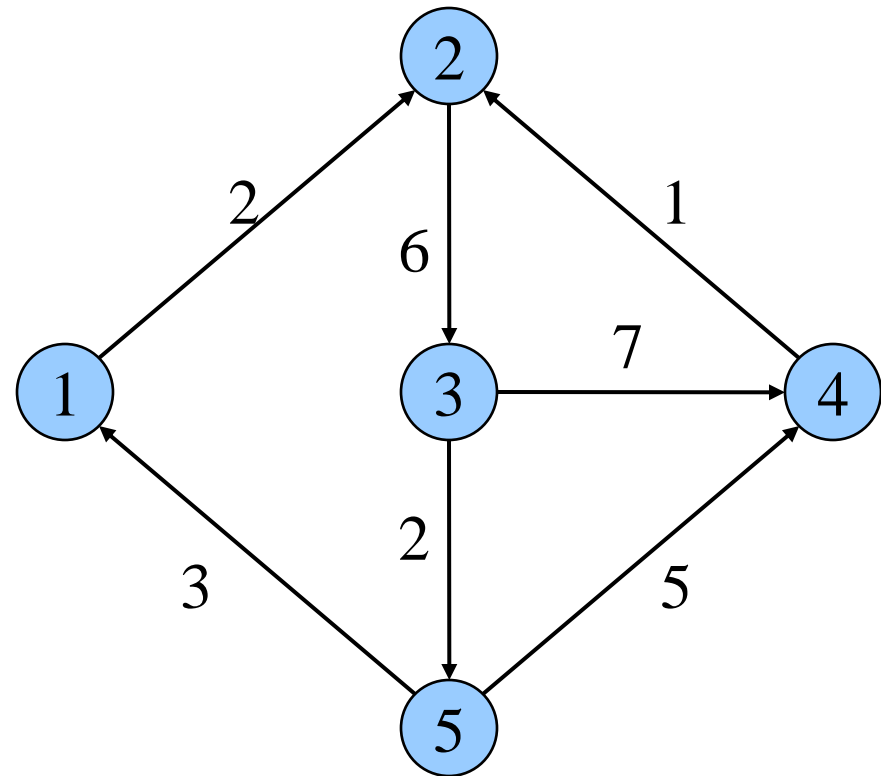


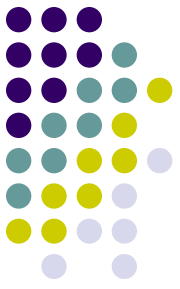


Korteste vei - alle-til-alle

- Eksempel:

$$l^{(3)} = \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} 0 \\ 11 \\ 5 \\ - \\ 3 \end{array} \begin{array}{c} 2 \\ 0 \\ 7 \\ 1 \\ 5 \end{array} \begin{array}{c} 8 \\ 6 \\ 0 \\ 7 \\ 11 \end{array} \begin{array}{c} 15 \\ 13 \\ 7 \\ 0 \\ 5 \end{array} \begin{array}{c} 10 \\ 8 \\ 2 \\ 9 \\ 0 \end{array}$$





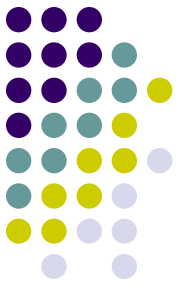
Korteste vei - alle-til-alle

- Kjøretid:
 - For alle mulige lengder:
 - Sjekk for alle par om det er bedre å gå innom k



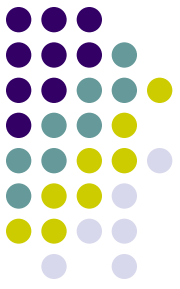
Korteste vei - alle-til-alle

- Kjøretid:
 - **For alle mulige lengder:**
 - Sjekk for alle par om det er bedre å gå innom k
 - $O(V * \dots)$



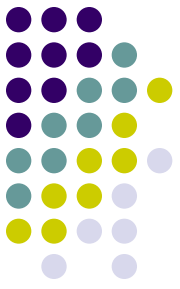
Korteste vei - alle-til-alle

- Kjøretid:
 - For alle mulige lengder:
 - **Sjekk for alle par** om det er bedre å gå innom k
 - $O(V * V^2 * \dots)$



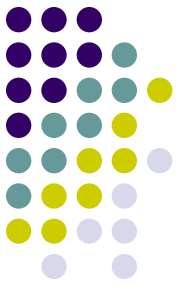
Korteste vei - alle-til-alle

- Kjøretid:
 - For alle mulige lengder:
 - Sjekk for alle par **om det er bedre å gå innom k**
 - $O(V * V^2 * V)$



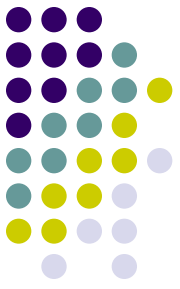
Korteste vei - alle-til-alle

- Kjøretid:
 - For alle mulige lengder:
 - Sjekk for alle par om det er bedre å gå innom k
 - $O(V^4)$



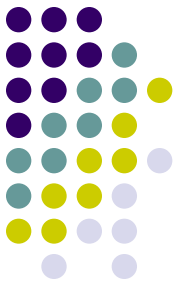
Korteste vei - alle-til-alle

- Men; hvorfor må vi ta en vei av lengde $m-1 + 1$, kan vi ikke ta veier av lengde $m/2 + m/2$?



Korteste vei - alle-til-alle

- Men; hvorfor må vi ta en vei av lengde $m-1 + 1$, kan vi ikke ta veier av lengde $m/2 + m/2$?
- Jo, det kan vi:



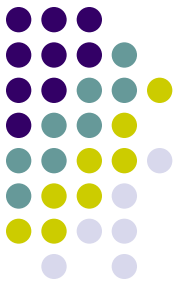
Korteste vei - alle-til-alle

- Men; hvorfor må vi ta en vei av lengde $m-1 + 1$, kan vi ikke ta veier av lengde $m/2 + m/2$?
- Jo, det kan vi:
 - Vi dobler da lengden på de korteste veiene hver gang, så trenger bare sjekke $\lg V$ lengder:



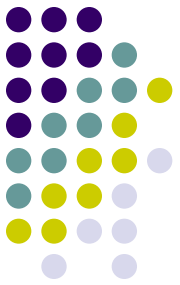
Korteste vei - alle-til-alle

- Men; hvorfor må vi ta en vei av lengde $m-1 + 1$, kan vi ikke ta veier av lengde $m/2 + m/2$?
- Jo, det kan vi:
 - Vi dobler da lengden på de korteste veiene hver gang, så trenger bare sjekke $\lg V$ lengder:
 - Kjøretid: $O(V^3 \lg V)$



Korteste vei - alle-til-alle

- Men; hvorfor må vi ta en vei av lengde $m-1 + 1$, kan vi ikke ta veier av lengde $m/2 + m/2$?
- Jo, det kan vi:
 - Vi dobler da lengden på de korteste veiene hver gang, så trenger bare sjekke $\lg V$ lengder:
 - Kjøretid: $O(V^3 \lg V)$
 - Men, fortsatt for dårlig



Korteste vei - alle-til-alle

Kode - eksempel



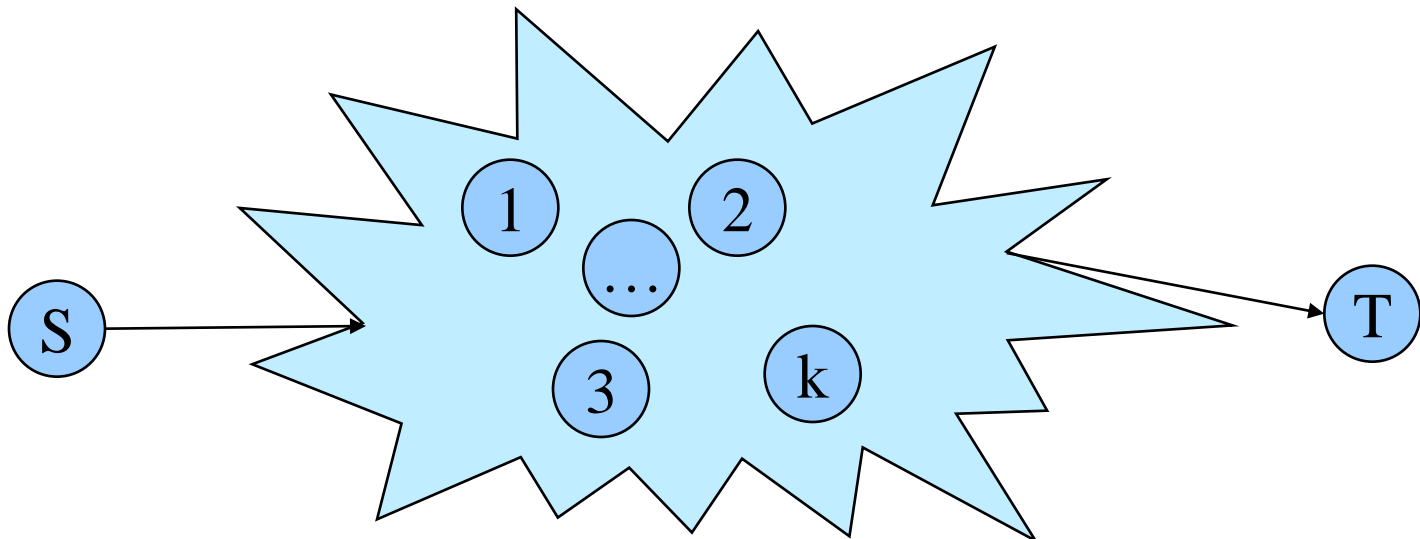
Floyd-Warshall

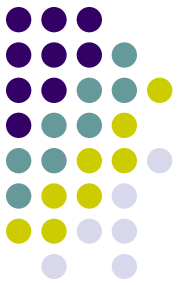
- Korteste vei alle-til-alle
- Bygger også på at en korteste vei kan dekomponeres



Floyd-Warshall

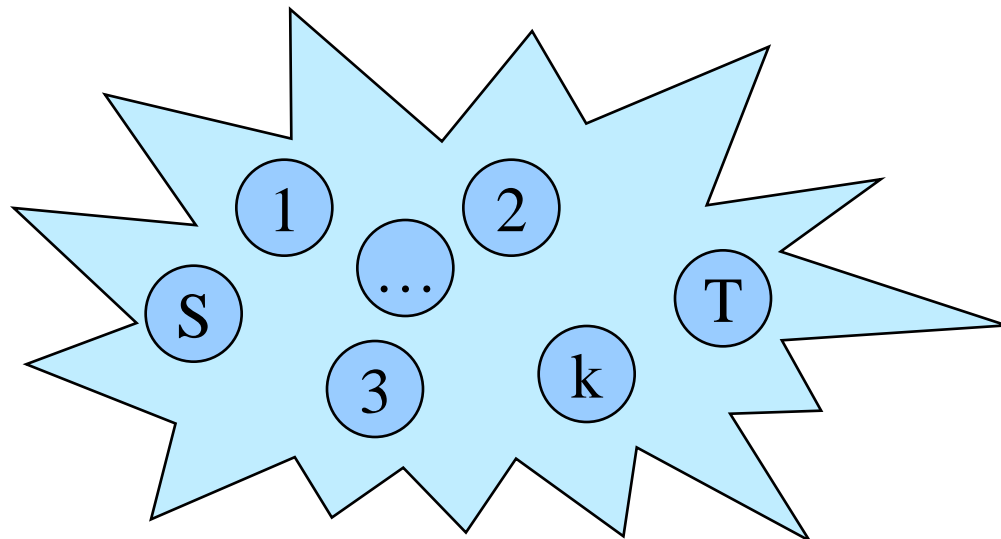
- La oss for øyeblikket kun se på korteste veier som bruker node $(1, 2, 3, \dots, k)$ foruten endenodene:

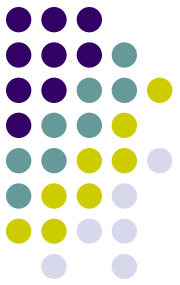




Floyd-Warshall

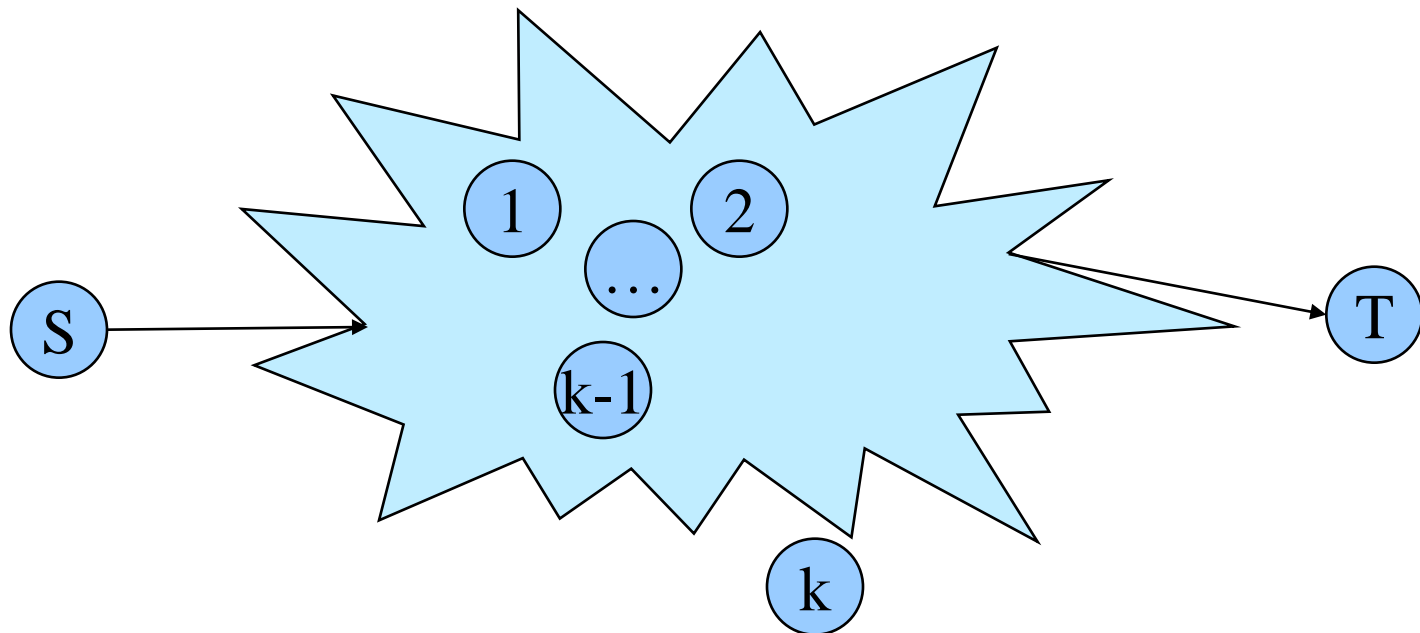
- La oss for øyeblikket kun se på korteste veier som bruker node $(1, 2, 3, \dots, k)$ foruten endenodene:





Floyd-Warshall

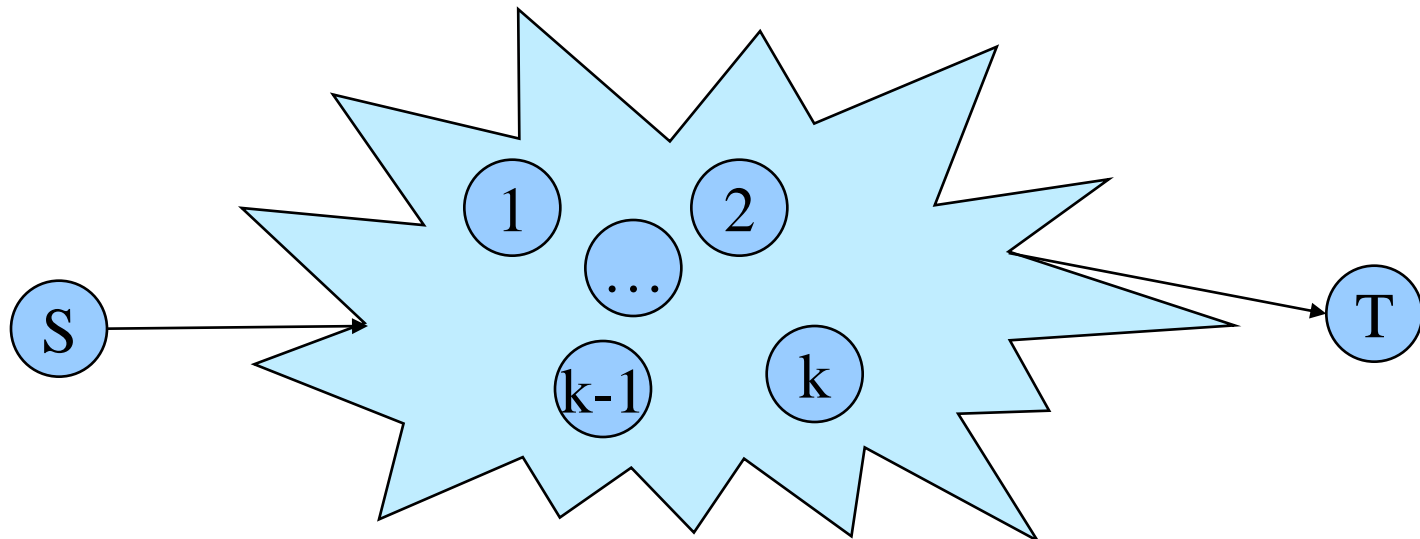
- Hver korteste vei vil da enten bestå av noder til-og-med $k-1$,





Floyd-Warshall

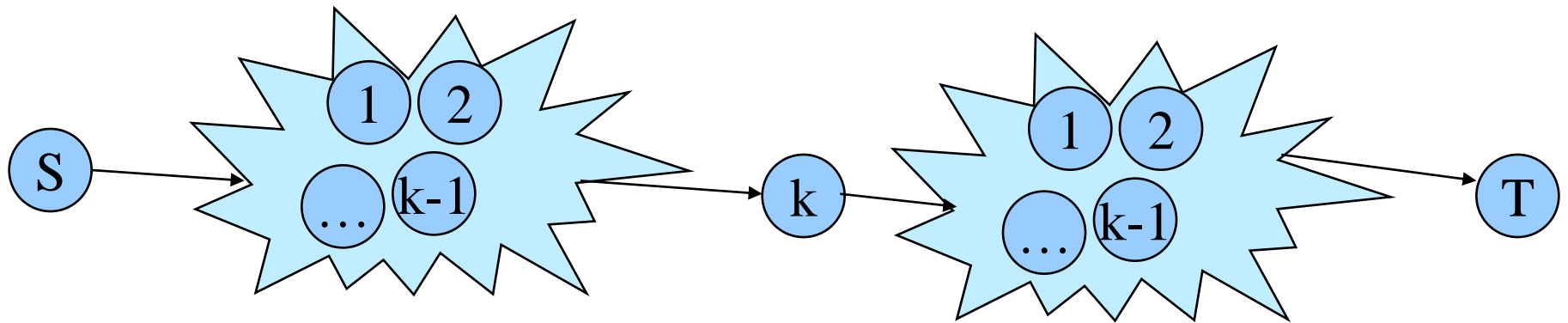
- Eller en korteste vei med k
 - Denne kan dekomponeres:





Floyd-Warshall

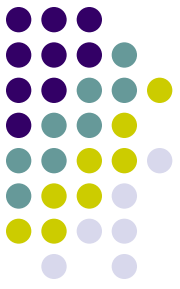
- Eller en korteste vei med k
 - Denne kan dekomponeres:





Floyd-Warshall

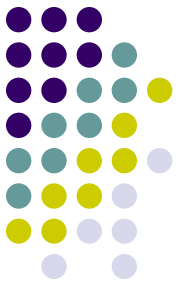
- Vi får da at korteste veier som bruker noder til og med node k , kan bygges opp av korteste veier som bruker til og med node $k-1$



Floyd-Warshall

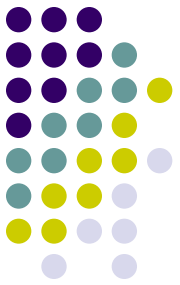
- Vi får da at korteste veier som bruker noder til og med node k , kan bygges opp av korteste veier som bruker til og med node $k-1$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$



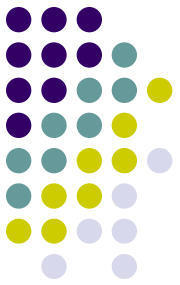
Floyd-Warshall

- Korteste veier som bruker til og med node V , er de korteste veiene i grafen



Floyd-Warshall

```
for k in xrange(v):  
    for i in xrange(v):  
        for j in xrange(v):  
            d[i][j] = min(d[i][j], d[i][k] + d[k][j])
```



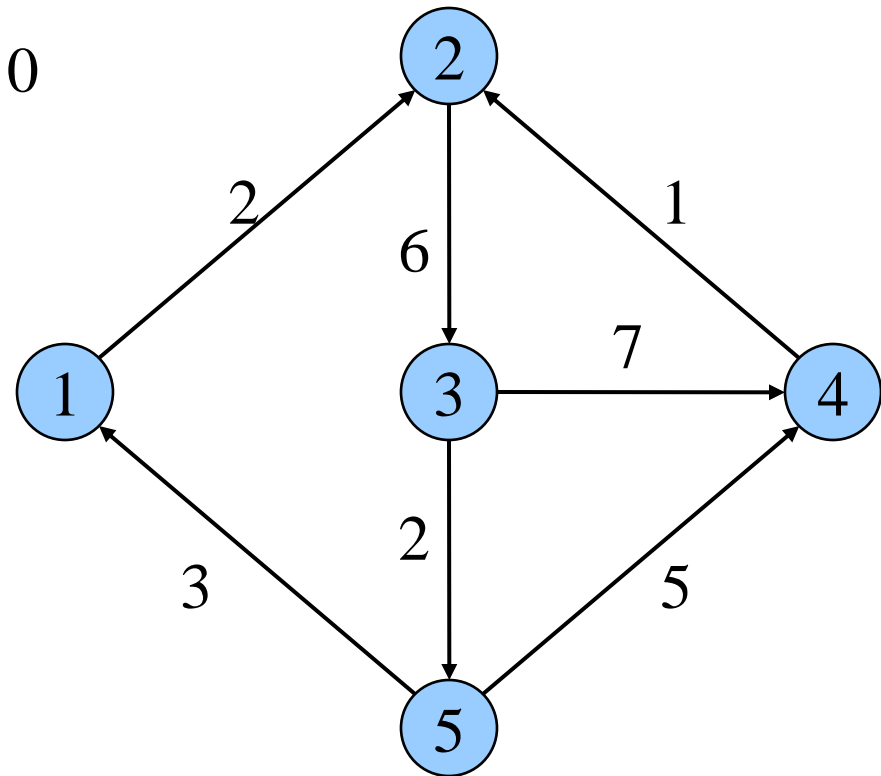
Floyd-Warshall

- Eksempel:

$d^{(0)} =$

	1	2	3	4	5
1	0	2	-	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	-	0	-
5	3	-	-	5	0

$k = 0$





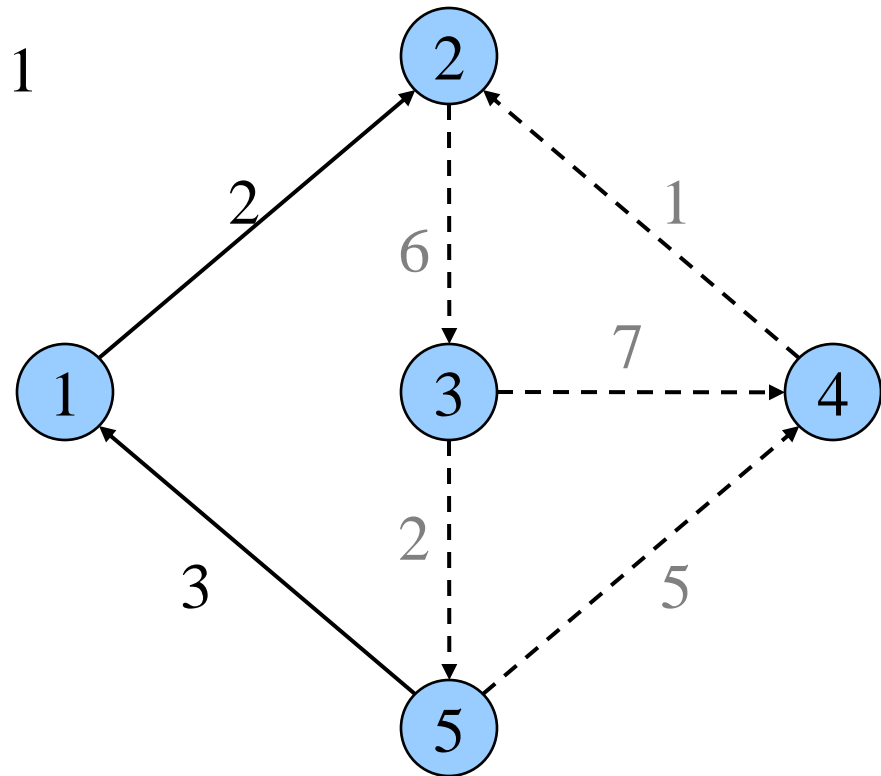
Floyd-Warshall

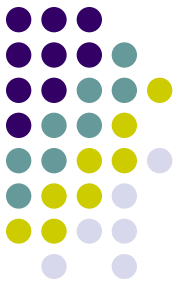
- Eksempel:

$d^{(1)} =$

	1	2	3	4	5
1	0	-	-	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	-	0	-
5	3	-	-	5	0

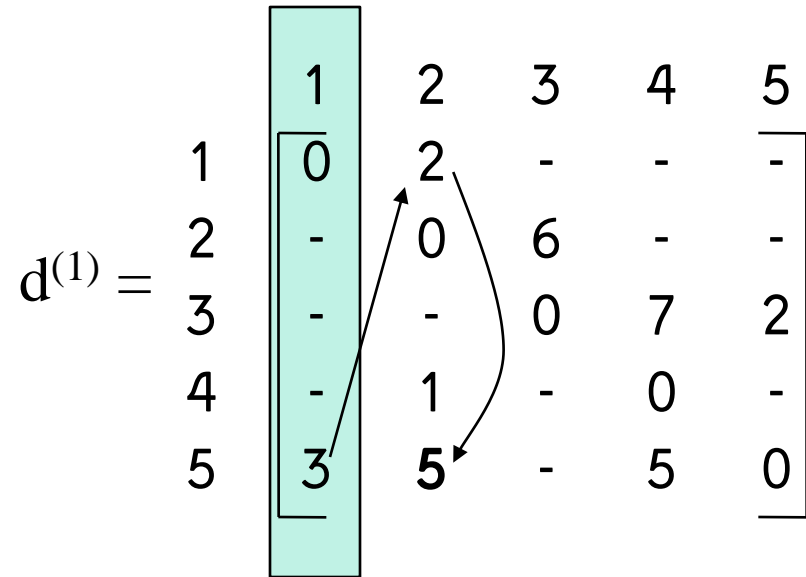
$k = 1$



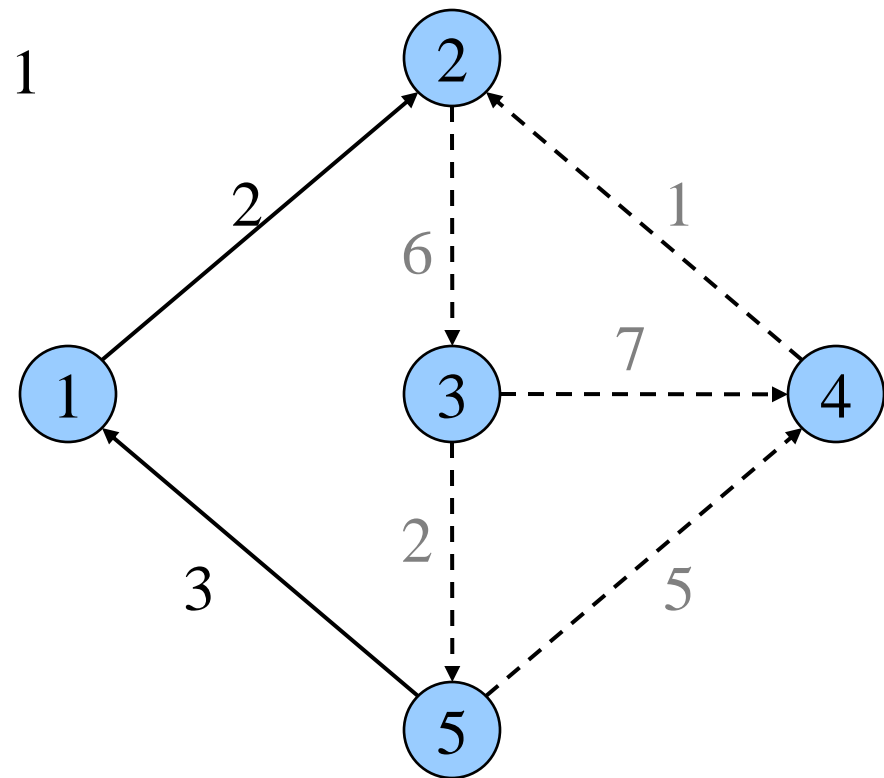


Floyd-Warshall

- Eksempel:



$k = 1$



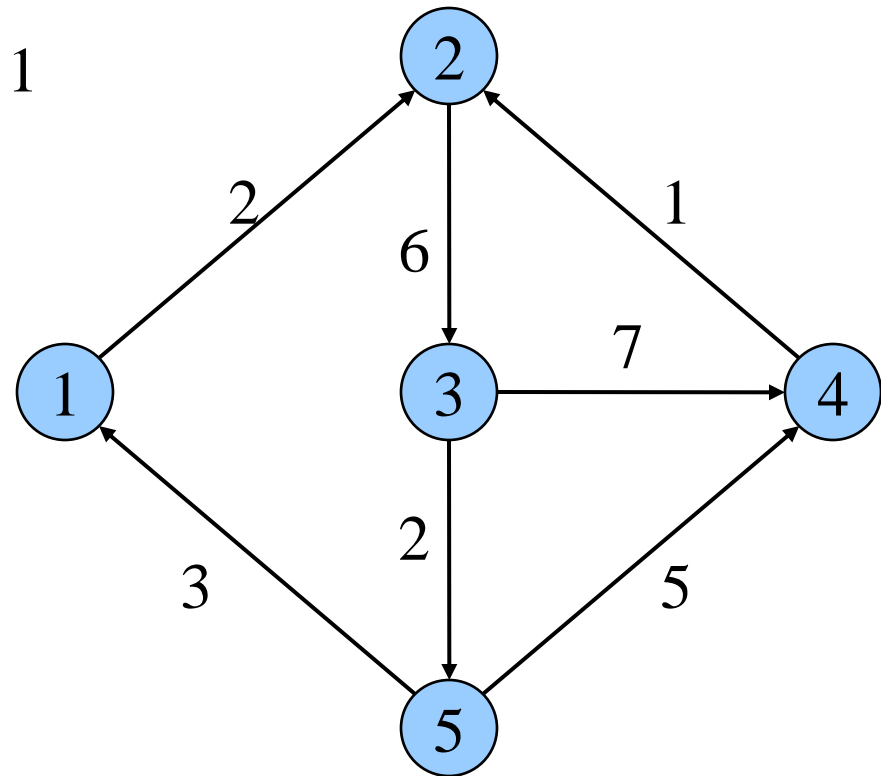


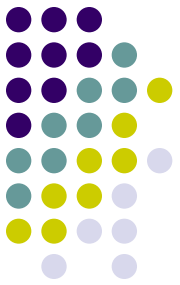
Floyd-Warshall

- Eksempel:

$$d^{(1)} = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 2 & - & - & - \\ - & 0 & 6 & - & - \\ - & - & 0 & 7 & 2 \\ - & 1 & - & 0 & - \\ 3 & 5 & - & 5 & 0 \end{bmatrix} \end{matrix}$$

k = 1





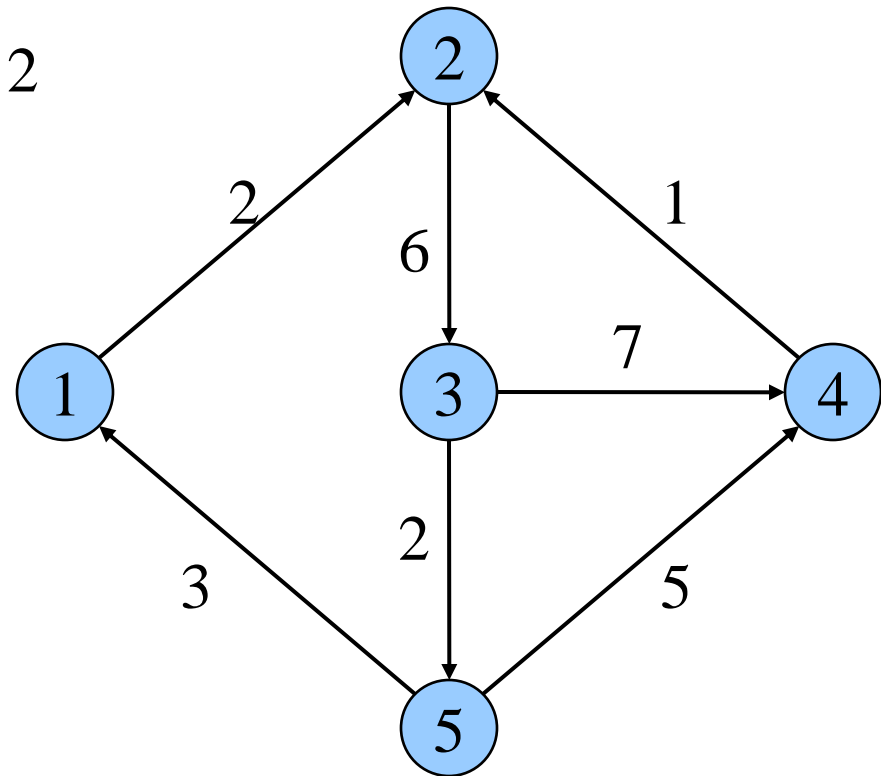
Floyd-Warshall

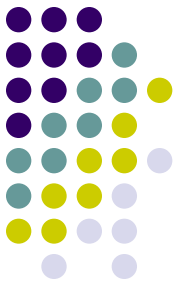
- Eksempel:

$d^{(2)} =$

	1	2	3	4	5
1	0	2	-	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	-	0	-
5	3	5	-	5	0

$k = 2$





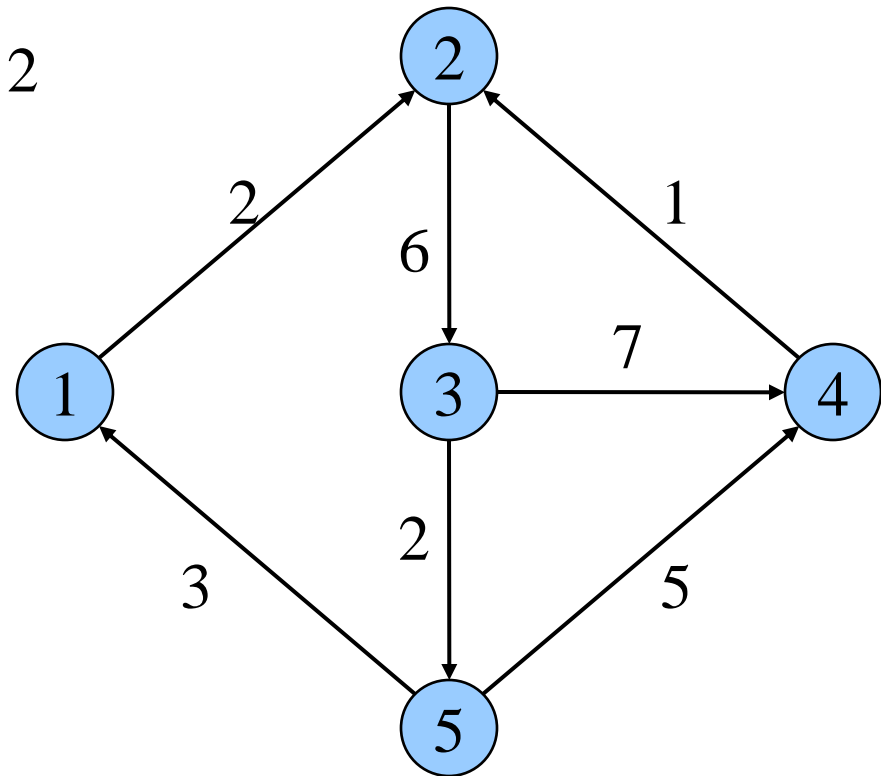
Floyd-Warshall

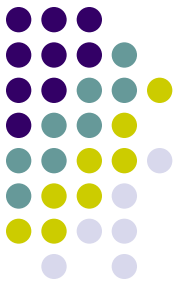
- Eksempel:

$d^{(2)} =$

	1	2	3	4	5
1	0	2	8	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	-	0	-
5	3	5	-	5	0

$k = 2$





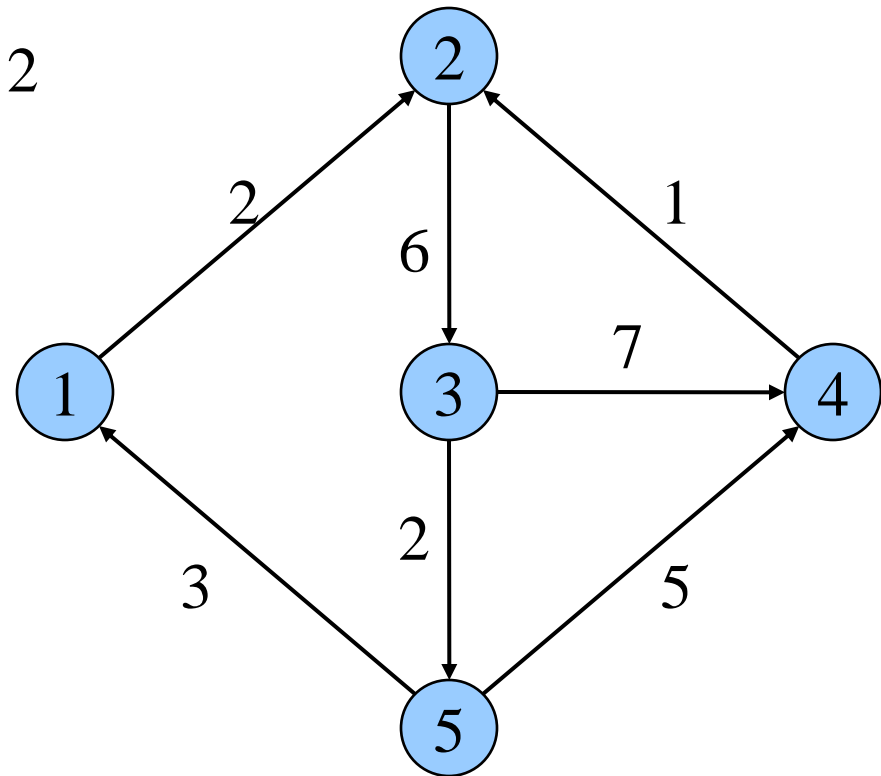
Floyd-Warshall

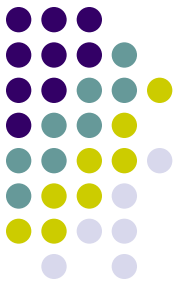
- Eksempel:

$d^{(2)} =$

	1	2	3	4	5
1	0	2	8	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	7	0	-
5	3	5	-	5	0

$k = 2$





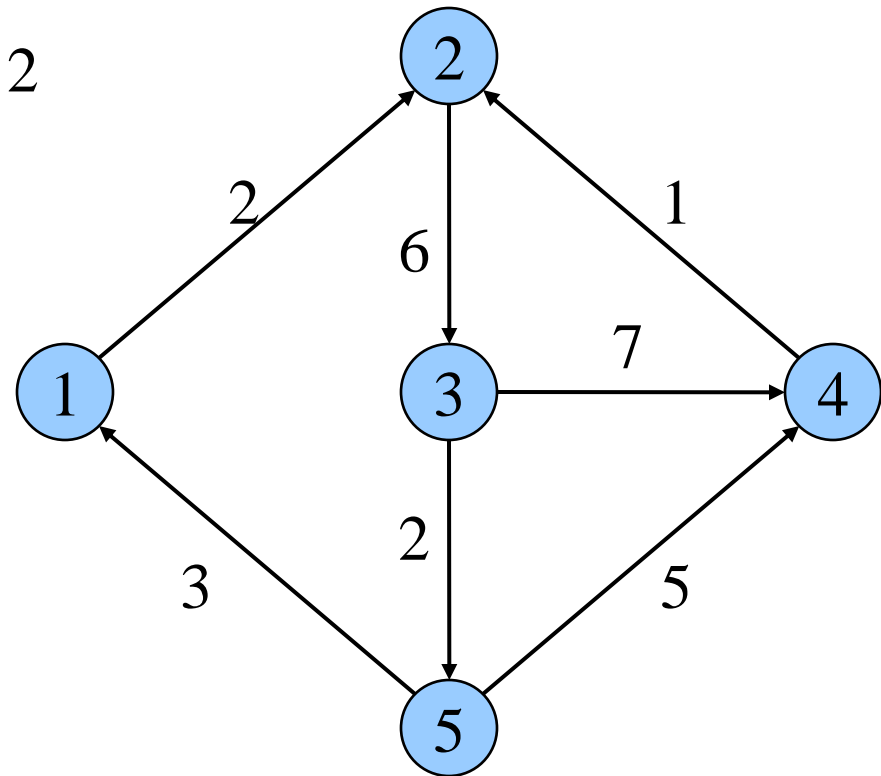
Floyd-Warshall

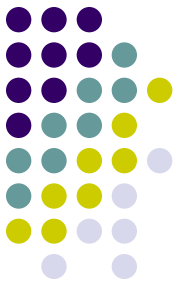
- Eksempel:

$d^{(2)} =$

	1	2	3	4	5
1	0	2	8	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	7	0	-
5	3	5	11	5	0

$k = 2$





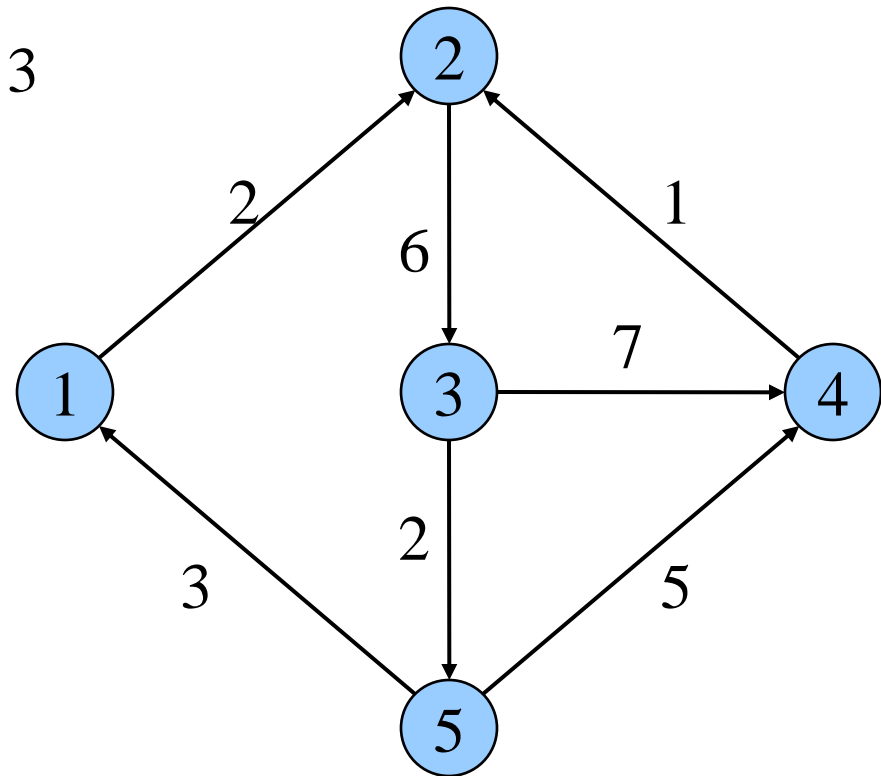
Floyd-Warshall

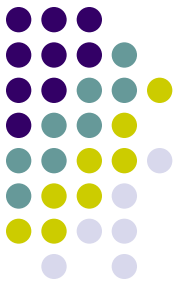
- Eksempel:

$d^{(3)} =$

	1	2	3	4	5
1	0	2	8	-	-
2	-	0	6	-	-
3	-	-	0	7	2
4	-	1	7	0	-
5	3	5	11	5	0

$k = 3$





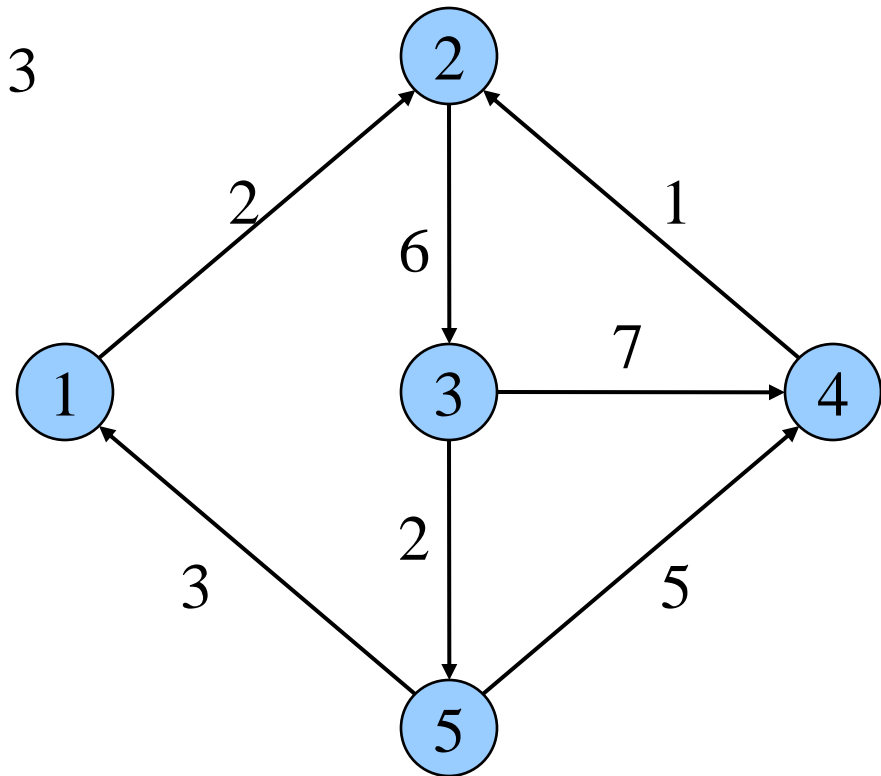
Floyd-Warshall

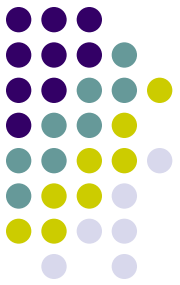
- Eksempel:

$k = 3$

$d^{(3)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	-	0	6	13	8
3	-	-	0	7	2
4	-	1	7	0	9
5	3	5	11	5	0





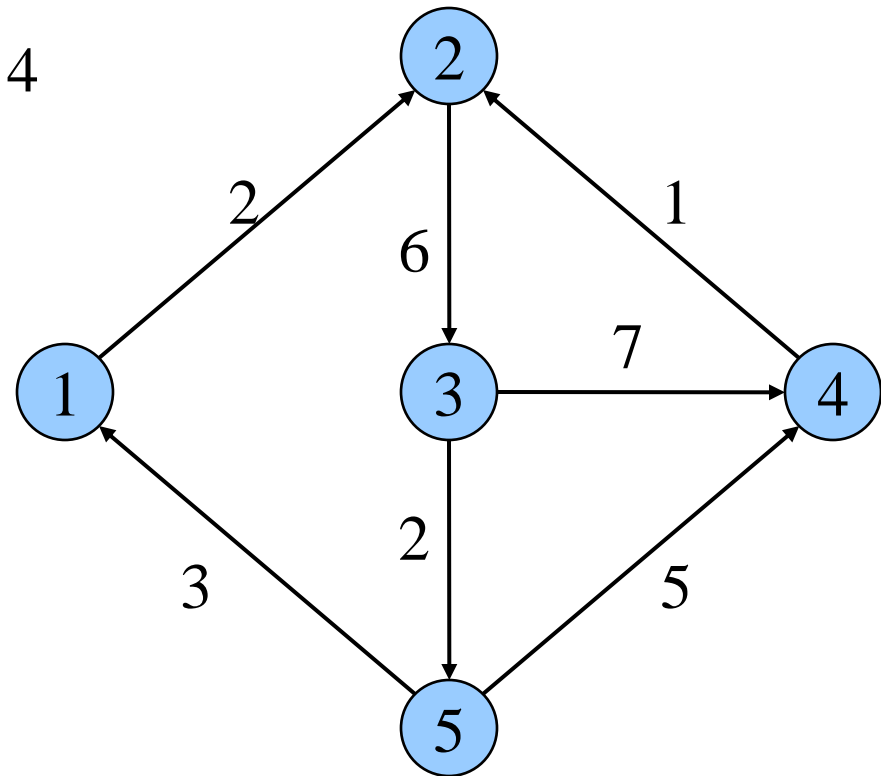
Floyd-Warshall

- Eksempel:

$k = 4$

$d^{(4)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	-	0	6	13	8
3	-	-	0	7	2
4	-	1	7	0	9
5	3	5	11	5	0





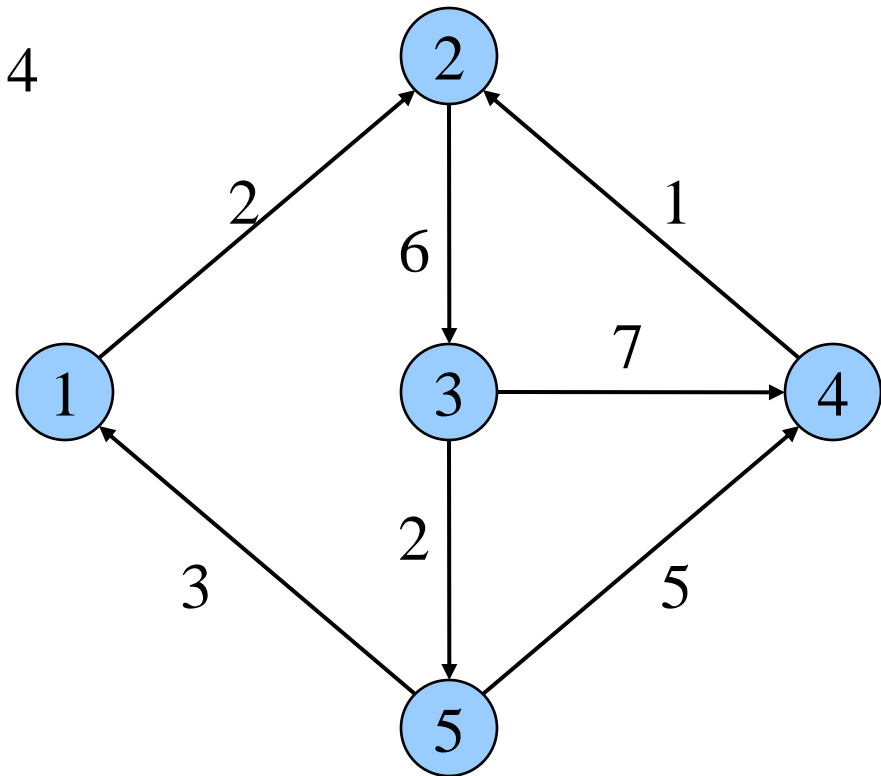
Floyd-Warshall

- Eksempel:

$k = 4$

$d^{(4)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	-	0	6	13	8
3	-	8	0	7	2
4	-	1	7	0	9
5	3	5	11	5	0





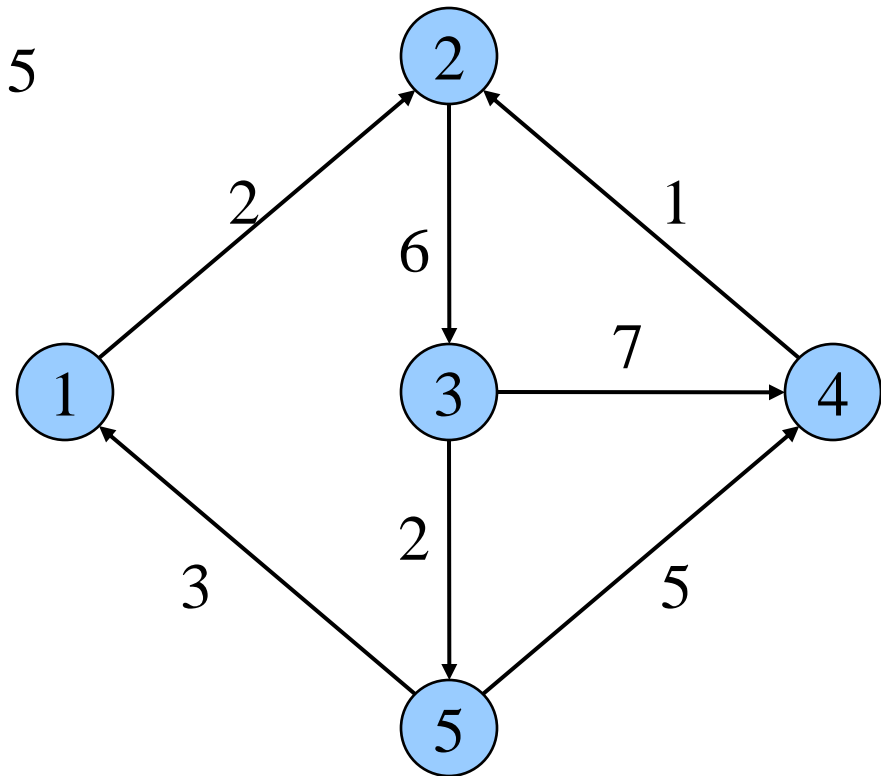
Floyd-Warshall

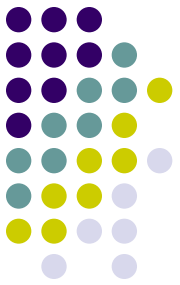
- Eksempel:

$k = 5$

$d^{(5)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	-	0	6	13	8
3	-	8	0	7	2
4	-	1	7	0	9
5	3	5	11	5	0





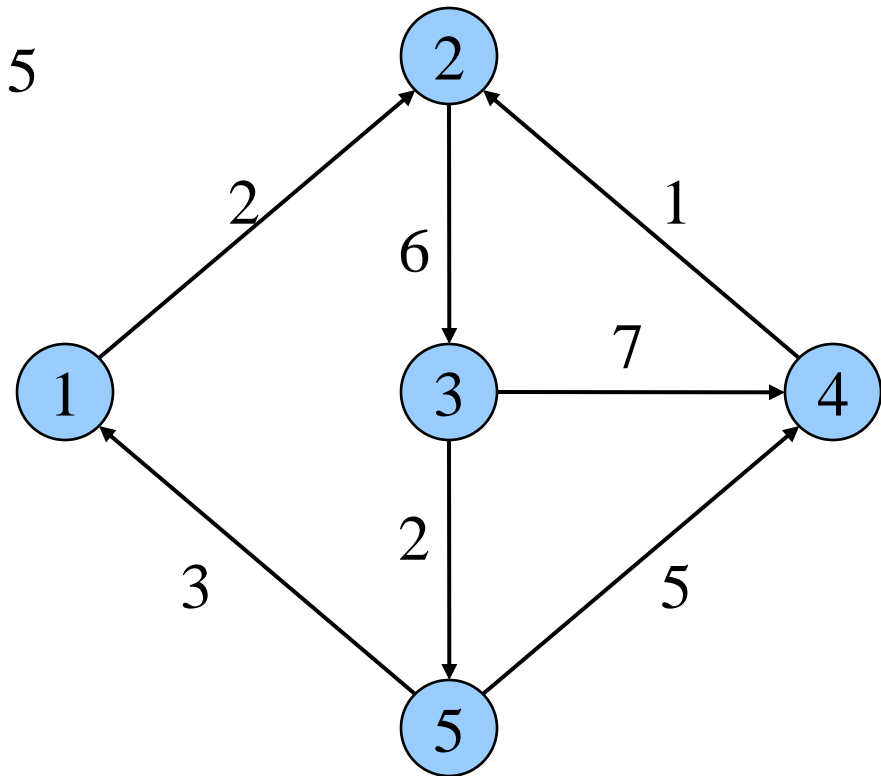
Floyd-Warshall

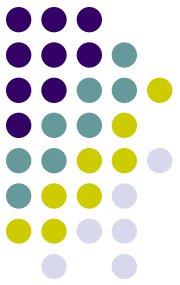
- Eksempel:

k = 5

$d^{(5)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	11	0	6	13	8
3	5	7	0	7	2
4	12	1	7	0	9
5	3	5	11	5	0





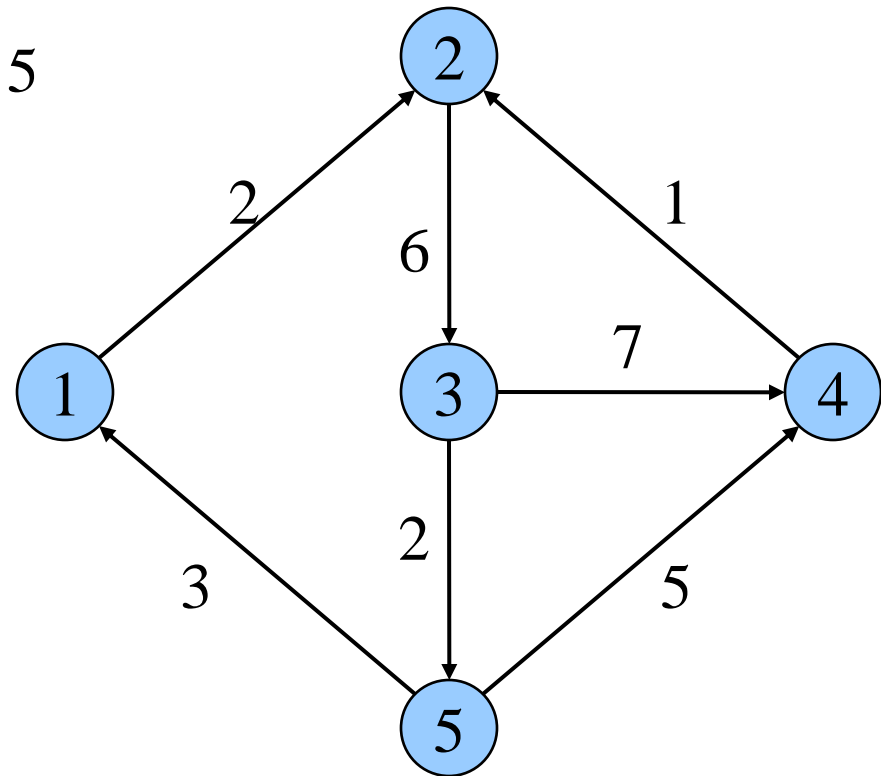
Floyd-Warshall

- Eksempel:

$k = 5$

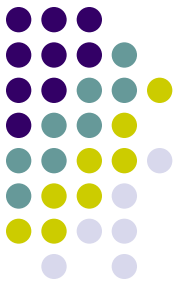
$d^{(5)} =$

	1	2	3	4	5
1	0	2	8	15	10
2	11	0	6	13	8
3	5	7	0	7	2
4	12	1	7	0	9
5	3	5	11	5	0



Floyd-Warshall

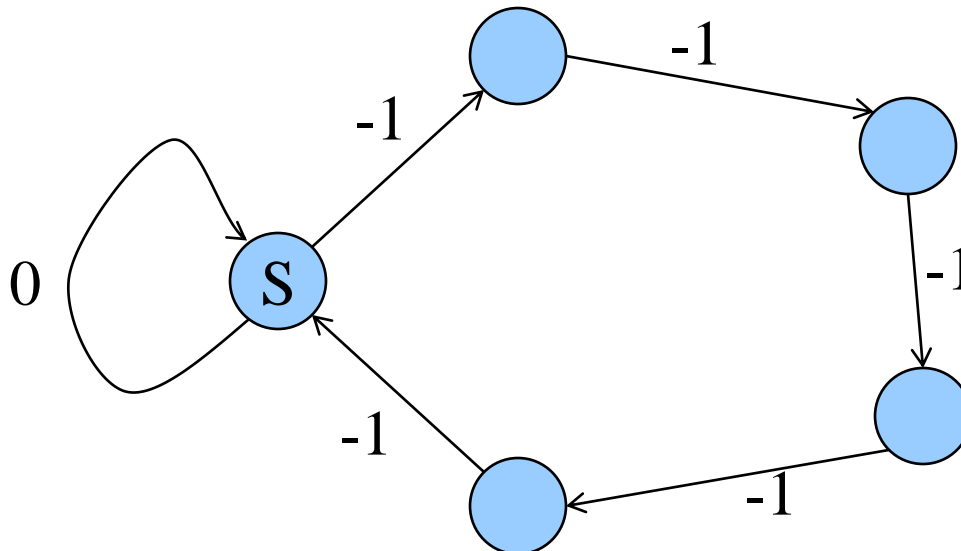
Kode - eksempel

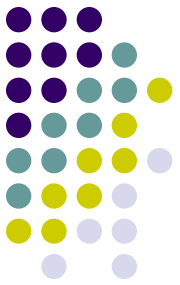




Floyd-Warshall

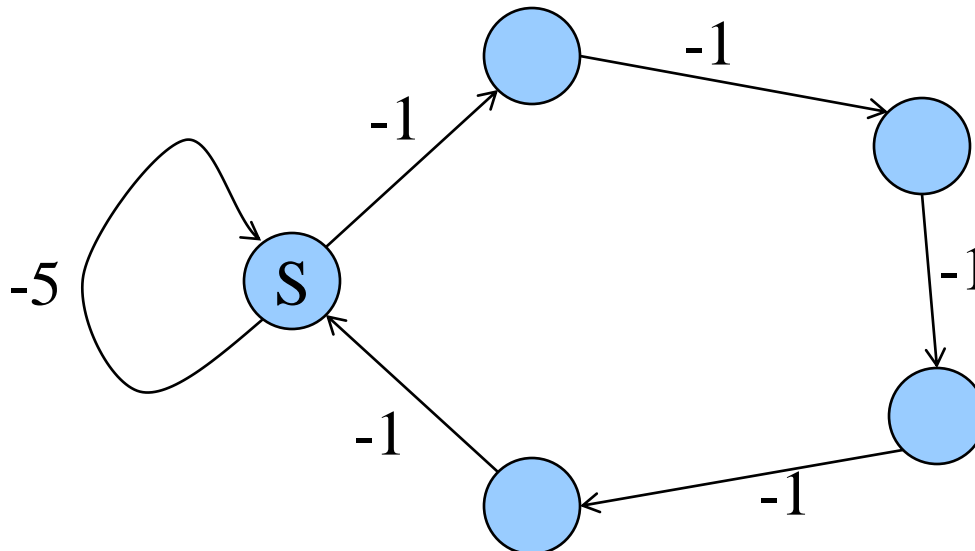
- Negative kanter:
 - Har vi en negativ sykel, vil en node få kostnad til seg selv lavere enn 0





Floyd-Warshall

- Negative kanter:
 - Har vi en negativ sykel, vil en node få kostnad til seg selv lavere enn 0





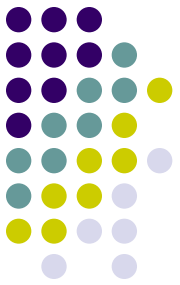
Floyd-Warshall

- Negative kanter:
 - Har vi en negativ sykel, vil en node få kostnad til seg selv lavere enn 0
 - Vi kan oppdage negative sykler ved at et element langs diagonalen blir mindre enn 0



Floyd-Warshall

- Kjøretid:
 - Tre for løkker til V
 - $O(V^3)$
 - Samme som Dijkstra, men takler negative kanter, og har lave konstante faktorer



Dynamisk programmering

- Metodene vi har sett på nå er eksempler på dynamisk programmering
- Vi uttrykker da den endelige løsningen som en funksjon av del-løsninger



Oppsummering

- Bellman-Ford:
 - Sjekk om vi kan få en bedre vei ved å bruke en kant til
- Korteste vei alle-til-alle:
 - Sjekk for alle par om det finnes en node som gir en bedre vei om vi går innom
- Floyd-Warshall:
 - Bruk de første k nodene
 - Sjekk for alle k om det er bedre å bruke k

Teoriøving 6



Teoriøving 7



Praksisøving 7 - Mumien

